Les Cahiers du GERAD ISSN: 0711–2440

Optimal picking policies for e-commerce warehouses

M. Schiffer, N. Boysen,
P. Klein, G. Laporte, M. Pavone

G–2019–62

August 2019

GERAD HEC Montréal
3000, chemin de la Côte-Sainte-Catherine
Montréal (Québec) Canada H3T 2A7

Tél. : 514 340-6053
Téléc. : 514 340-5665
info@gerad.ca
www.gerad.ca

Optimal picking policies for e-commerce warehouses

Maximilian Schiffer [a,b]
Nils Boysen [c]
Patrick Klein [b]
Gilbert Laporte [a,d]
Marco Pavone [e]

[a] *GERAD, Montréal (Québec), Canada, H3T 2A7*

[b] *TUM School of Management, Technical University of Munich, Munich, Germany, 80333*

[c] *Friedrich Schiller University Jena, Jena, Germany, 07743*

[d] *CIRRELT and Canada Research Chair in Distribution Management, HEC Montréal, Montréal (Québec), Canada, H3T 2A7*

[e] *Stanford University, Stanford CA 94305, USA*

schiffer@tum.de
nils.boysen@uni-jena.de
patrick.sean.klein@tum.de
gilbert.laporte@cirrelt.ca
pavone@stanford.edu

August 2019
Les Cahiers du GERAD
G–2019–62

Abstract: In e-commerce warehouses, online retailers increase their efficiency by using a mixed-shelves (or scattered storage) concept, where unit loads are purposefully broken down into single items, which are individually stored in multiple locations. Irrespective of the stock keeping units a customer jointly orders, this storage strategy increases the likelihood that somewhere in the warehouse the items of the requested stock keeping units will be in close vicinity, which may significantly reduce an order picker's unproductive walking time. This paper optimizes picker routing through such mixed-shelves warehouses. Specifically, we introduce a generic exact algorithmic framework that covers a multitude of picking policies, independently of the underlying picking zone layout, and is suitable for real-time applications. This framework embeds a bidirectional layered graph algorithm which provides the best known performance for the simple picking problem with a single depot and no further attributes. We compare three different real-world e-commerce warehouse settings that differ slightly in their application of scattered storage and in their picking policies. Based on these, we derive additional layouts and settings that yield further managerial insights. Our results reveal that the right combination of drop-off points, dynamic batching, the utilization of picking carts, and the picking zone layout can greatly improve the picking performance. In particular, some combinations of policies yield efficiency increases of more than 20% compared with standard policies currently used in practice.

Keywords: Mixed-shelves warehouse, order picking policies, picking zone layout, dynamic programming

1 Introduction

A paradigm change in the retail business, caused by rapidly growing e-commerce (Statista, 2017; Cui et al., 2019), makes today's warehouses the focus of efficient operations in business-to-customer (B2C) markets. While warehouses have traditionally been viewed as remote, negligible, and cost-efficient planning components, today's warehouses have evolved into technology-enriched logistics facilities that play a key role in retail supply chains. For companies of any significant size, the warehouse of today and of the future performs several of the functions that have previously been managed by conventional stores. Consequently, the attention of firms to innovative concepts and efficient warehouse operations is steadily increasing, as warehouses are regarded as a key success factor in the highly competitive e-commerce market. In these warehouses, efficient daily operations are central to running a profitable e-commerce business and to fulfilling customer expectations such as same-day deliveries. In this context, efficiently routing pickers through warehouses constitutes a crucial determinant of profitability.

In e-commerce warehouses, two fundamentally different concepts dominate: robotized parts-to-picker warehouses, where (KIVA) robots bring man-high shelves to stationary pickers (Azadeh et al., 2018), and human operated picker-to-parts warehouses, in which pickers traditionally walk through the shelves and collect stock keeping units (SKUs). These two concepts entail a trade-off between investment costs, scalability, and efficiency.

Parts-to-picker warehouses have high investment costs that increase significantly with the number of robots, but are highly efficient when operated at their maximum throughput. They are hardly scalable to rare events during which the workload increases significantly, e.g., on Black Friday or for Cyber Monday sales.

Picker-to-parts warehouses have low investment costs and are less efficient than robotized systems. In contrast, these warehouses allow—by hiring additional temporary staff—the necessary flexibility in peak times. Hence, major players in the industry agree that picker-to-parts warehouses should remain in parallel to robotized warehouses in order to provide flexibility.

With standardized third-party robotized warehouse solutions across companies, increasing the efficiency of picker-to-parts warehouses becomes even more crucial for e-commerce retailers who vie to derive a competitive advantage. In traditional picker-to-parts warehouses (with unit loads kept together) the storage assignment plans result in excessive unproductive walking for the pickers, corresponding to as much as 50% of their working hours (de Koster et al., 2007). To increase the efficiency of such warehouses, many online retailers in the B2C segment apply the mixed-shelves or scattered storage concept, see Figure 1 (Weidinger and Boysen, 2018). Under this storage assignment policy,

Figure 1: Mixed-shelves in a scattered storage e-commerce warehouse.[1]

incoming loads of SKUs are purposefully broken down into single items scattered over all parts of the warehouse, thus increasing the likelihood that, given the volatile demands of the final customers, items finally ordered together will be in close vicinity somewhere in the warehouse (Boysen et al., 2019a). However, competitive routing algorithms that determine picker paths to collect SKUs remain crucial to an efficient exploitation of the scattered storage in e-commerce warehouses, and to the reduction of unproductive picker walking time. In particular, large e-commerce warehouses subdivide their order fulfillment process into three stages, i.e., order picking, intermediate storing of picked bins, and order accumulation and packing (see Section 2.1). While the latter two stages are rather standardized and can to a large extent be automated, order picking remains the most crucial and labor intensive stage and is therefore the focus of this paper.

Today's e-commerce warehouses are divided into picking zones, i.e., limited blocks of shelves in which single pickers operate (Figure 2). These zones have a rectilinear layout, which means that the pickers always move according to a Manhattan metric, with shelves placed along parallel aisles and one or several drop-off points at which pickers can hand over items. Online retailers apply batch picking, i.e., they unify multiple picking orders into a single pick list (Boysen et al., 2019a) which is then processed by a picker in a dedicated zone. The picker starts with a cart and a pick list at a drop-off point and returns to it once the pick list is completed. Despite the regular rectilinear layout and the rather simple picking process, picking zones may differ in terms of the number of cross-aisles and aisles they contain. Further, although most companies rely on the scattered storage principle and agree on efficient picking as a key to successful operations, there exists no consensus on which general attributes a good picking policy and a picking zone layout should possess. In this paper we will focus on four main attributes which have been identified as the most frequently implemented in practice in the recent survey paper of Boysen et al. (2019a):

i) *a variable multi-block layout* which differs in terms of the number of aisles and cross-aisles;
ii) *multiple drop-off points* which increase the number of points in a picking zone where completed bins can be handed over to a central conveyor system;
iii) *dynamic batching policies* which allow processing several pick lists in parallel by equipping the picking cart with multiple bins (Weidinger et al., 2019); and
iv) *cartless subtours* which allow pickers to intermediately park their clumsy cart in order to pick a few items on a subtour much faster.

In addition, several site visits we have made in European e-commerce warehouses have confirmed that these policies and layout options are applied in various combinations (see Section 2.2), and benefits can be achieved through optimization. However, performance gains come at the price of additional investments which may differ for each individual setting. Hence, we evaluate the impact on performance of the picking policy and layout options which allows us to quantify the improvement potential of each measure without relating it to varying and hence hardly assessable investment costs. To set our study apart from recent research, we first briefly review the related literature in Section 1.1, before further detailing the aims and scope of our work in Section 1.2.

1.1 Literature review

Efficient order processing in warehouses and distribution centers has been vividly discussed in recent years. For an overview of the rich body of literature on traditional picker-to-parts warehouses, modern robotized warehouses, and the peculiarities of warehousing in the e-commerce era, we refer the interested reader to the surveys of de Koster et al. (2007); Azadeh et al. (2018), and Boysen et al. (2019a). In the following, we only survey the picker routing literature related to the planning tasks outlined above.

[1]The picture is published under the Creative Commons License (BY 2.0). Its author is Álvaro Ibáñez.

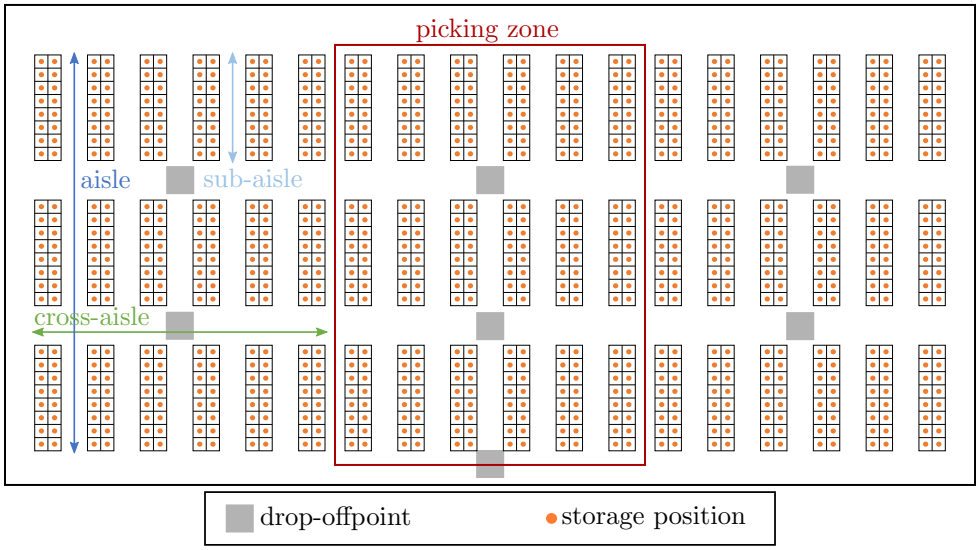


Figure 2: Example of the storage area of an e-commerce warehouse and a picking zone.

Picker routing starting and ending at a central drop-off point corresponds to the traveling salesman problem (TSP). In their seminal paper, Ratliff and Rosenthal (1983) showed that in single-block warehouses consisting of parallel aisles with cross-aisles at the front and back, the particular structure of the distance matrix allows solving the resulting TSP in polynomial time by dynamic programming. Roodbergen and de Koster (2001) extended this approach to two-block warehouses with an additional middle aisle. Another extension of de Koster and van der Poort (1998) considers a handover of complete orders at the start and end of each aisle. This is relevant if bins can be placed on conveyors leading along the cross-aisles of a single block, but does not account for multiple drop-off points in a multi-block warehouse. Note that a conveyor directly installed within a cross-aisle is an obstacle for pickers, so that multiple drop-off points giving access to conveyors installed under the ceiling seem preferable. Recently, Pansart et al. (2018) generalized the approach of Ratliff and Rosenthal (1983) to multi-block warehouses using the polynomial-time algorithm of Cambazard and Catusse (2018) for the rectilinear Steiner tree problem in the plane. As can be seen, exact algorithms for the picker routing problem are rare and limited to special cases.

Several heuristics have been proposed for the picker routing problem itself and for some of its variants, e.g., picker routing combined with the selection of pick positions or with zoning and batching policies. For a rich overview of these heuristics, we refer the reader to de Koster et al. (2007). While fundamental work including performance analysis was carried out by Hall (1993) and Petersen (1997), de Koster and van der Poort (1998) provided comparisons between exact and heuristic algorithms for specific layouts. The more recent approximate algorithms adapt competitive TSP heuristics to the picker routing problem (Theys et al., 2010). Gue et al. (2006); Hong et al. (2012a), and Chen et al. (2013) have all focused on blocking aspects in narrow aisles. Given the dedicated zoning policy and layout of modern e-commerce warehouses, this blocking issue is negligible nowadays. Only a single publication (Goetschalckx and Ratliff, 1988) exists on pickers with different travel modes, i.e., pickers who are allowed to park their cart and perform cartless subtours on foot. However, that paper assumes that a picker moves the vehicle along a single aisle, but starting and stopping the vehicle to retrieve items on foot consumes additional time. As can be seen, no heuristic exists that captures the generic nature of our problem.

Weidinger et al. (2019) showed that the integrated selection of pick positions for a specific demand in a mixed-shelves warehouse makes the picker routing problem strongly NP-hard, even in elementary block-shaped warehouses with parallel aisles. Algorithms for this problem class were proposed by Daniels et al. (1998); Weidinger (2018), and Weidinger et al. (2019). These authors decompose the

problem by fixing the selection of pick positions and treat the second-stage routing problem heuristically. Overviews of zoning and batching policies can be found in de Koster et al. (2007) and Boysen et al. (2019a). Recent contributions to this field by Bozer and Kile (2008); Henn and Wäscher (2012); Hong et al. (2012b), and Zulj et al. (2018), have focused on heuristics for a static batching policy. In practice, the two planning tasks of selecting pick positions and batching and zoning are decoupled from the picker routing decision. Recently, Weidinger (2018) gave proof to this common practice and showed that solutions derived in this fashion yield optimality gaps well below one percent compared to the integrated planning approach. Hence, we exclude these problems from our real-world study, and we assume these decisions to have been taken at an upstream decision stage.

Concluding, no exact or heuristic algorithm exists that captures the generic nature of our problem, namely *i) a variable multi-block layout*, *ii) multiple drop-off points*, *iii) dynamic batching policies*, and *iv) cartless subtours*. In particular, requirements *ii)–iv)* have, to the best of our knowledge, not yet been incorporated in any exact or heuristic algorithm.

1.2 Aims and scope

In this paper we close the gap in the literature outlined in Section 1.1 by developing the first generic exact algorithm that can handle the four features just mentioned. Its computational requirements must be sufficiently low to allow for real-time application in practical settings. Given this, the methodological and managerial contribution of this paper is threefold. First, we develop an exact algorithm for picker routing capable of capturing the four main attributes identified by Boysen et al. (2019a). Second, we provide additional preprocessing techniques that allow the real-time use of this algorithm in real-world applications. Third, based on this methodological framework, we conduct different experiments in order to benchmark several picking policies and layout options on realistic data sets extracted from three corporate case studies.

Our algorithm provides a new state of the art for picker routing problems. In particular, we outperform the previous best known exact algorithm (Pansart et al., 2018) for single pick lists in multi-block warehouses significantly in terms of computational time, and we obtain a better scalability for large-size instances with up to 11 cross-aisles. In addition, our algorithm is significantly more general since it can solve picker routing problems with multiple pick lists, dynamic batching strategies, multiple drop-off points, and cartless subtours. We use this algorithm to identify the benefits of all considered picking policy design options by means of a full factorial experiment. Analyzing these results, we derive managerial insights into best practices for the design of picking policies and picking zone layouts in online retailing. Our study addresses operational aspects, e.g., the impact of parallel processing of multiple pick lists, but also strategic dimensions which affect the warehouse design, e.g., the impact of multiple drop-off points and the number of cross-aisles.

1.3 Organization of the paper

The remainder of this paper is structured as follows. Section 2 further details the fulfillment process in an e-commerce warehouse, details the settings of picking policies we have observed in practice, and provides a formal definition of the planning problems at hand. Section 3 develops our algorithm. Extensive computational results are presented in Section 4. Section 5 concludes this paper and provides several managerial insights.

For the sake of readability, we shift all technical definitions used in this paper to Appendix B. When introducing a defined term, we mark it with a dagger† . All proposition proofs are provided in Appendix C.

2 Problem description

This section introduces the planning problem. We first detail the order fulfillment process in e-commerce warehouses to show the integration of our planning problem into daily operations. We then discuss the settings of picking policies that we have observed in practice, before formally introducing the generalized order picking problem (GOPP).

2.1 Order fulfillment process

For the sake of conciseness we exclude the ingoing flow of the warehouse and focus on the outgoing flow. Once items are stored and scattered around the warehouse, the order fulfillment process in a mixed-shelves warehouse consists of three basic steps.

Order picking: First, customer orders must be picked from their respective storage positions. Human pickers, each equipped with a small picking cart that carries small bins, collect items into these bins. A picker receives empty bins and pick lists, each associated with a bin, at a drop-off point, strides (directed by a hand-held scanner or a pick-by-voice system) through the warehouse, collects the requested items from the shelves and puts them into their respective bin on the cart. Once the picker has completed one or multiple bins, she hands the bins over to the central conveyor system at a drop-off point and starts processing the next pick lists. This process is typically executed under a zoning and batching policy (de Koster et al., 2007). Instead of storing all items in a single monolithic area, online retailers subdivide their vast shop floors into multiple zones to allow for a parallel processing of customer orders. A picker operates exclusively in one of these zones and only picks the part of an order that is stored in her assigned zone. By parallelizing the picking process in this fashion, order pickers traverse smaller areas of the warehouses, so that their unproductive walking time is reduced. A further reduction of unproductive walking time can be achieved by unifying multiple orders into batches that are jointly retrieved on the picker's tour through her zone. This results in bins filled with partial orders for multiple customers.

Intermediate storing of picked bins: Once handed over to the central conveying system at a drop-off point, these bins are intermediately stored until all those belonging to the same wave of customer orders have arrived from all zones. Different storage devices exist. Some operators apply lane-based systems where bins of a wave are channeled in a conveyor queue. Others apply automated storage and retrieval systems, where bins are stored in crane-operated aisles of high-bay storage racks (Boysen et al., 2018). In smaller warehouses, loop-based systems where bins circle until a wave is complete may be applied (Gallien and Weber, 2010).

Order accumulation and packing: Finally, a wave of bins released from intermediate storage arrives in the order accumulation area. Here, the incoming items are sorted according to customer orders. Some operators apply so-called put walls for this task, where human logistics workers place items on small shelves, each temporarily dedicated to a specific customer order (Boysen et al., 2019b). On the other side of the wall, packers receive completed orders, pack them into their shipping cartons, and forward them via conveyors to their dedicated trailers. Other warehouses apply automated belt sorters, e.g., tilt tray or cross-belt sorters, where items are collected in packing stations arranged along the belt (Boysen et al., 2018).

Both *intermediate storing* and *packing* are process steps that can easily be standardized and organized efficiently. However, *order picking* includes unpredictable components because it depends on incoming customer orders and requires the largest fraction of the workforce. Therefore, an efficient routing algorithm plays a crucial role to ensuring efficient order fulfillments.

The inputs needed for the creation of route plans are the pick lists, a pick list sequence, and the assignment of SKUs to pick lists. Integrated problems that combine order batching or sequencing with picker routing have been discussed (Weidinger and Boysen, 2018), but while they provide challenging research questions, their solutions are less relevant in practice. Indeed, operators typically solve the

batching and sequencing problems at an upper level, decoupled from the picker routing for the following reason: Both problems heavily depend on the priorities of the incoming orders. Those orders with the closest due date of their associated outbound trucks are selected as the wave of orders to be processed next. Therefore, all orders of the current wave are urgent and there is no need to further differentiate between their priorities during order picking. Recent research has corroborated this common practice. Weidinger (2018) has shown that in practice selecting the shelves from which each item is to be picked in a scattered storage warehouse can be solved independently of the picker routing problem. Indeed, selecting shelves such that the warehousing area of the current pick list is minimized and handing these pick positions over to picker routing leads to near-optimal solutions with optimality gaps well below one percent (Weidinger, 2018).

2.2 Picking policies

In this paper, we develop a generalized exact algorithm for picker routing in e-commerce warehouses. We create different settings based on different order picking policies that we have observed in practice in mixed-shelves warehouses. We name these policies according to the companies at which we have observed them:

Amazon Europe: In Amazon's distribution centers in Bad Hersfeld (Germany) and Poznan (Poland) we have observed the following setting. Amazon uses multiple drop-off points to a central conveyor system to hand over completed bins. Pickers often pass by these local drop-off points during a tour, which offers the additional flexibility of dynamic batching. Instead of picking batch after batch with intermediate returns to a central drop-off point in a static manner, pickers can hand over just a subset of completed bins at a local drop-off point and retrieve new empty bins for successive orders. Thus, Amazon's picker routing procedure additionally considers the dynamic batching of an order set by inserting visits at different drop-off points in the tour where the picker hands over subsets of completed bins and starts handling new orders.

Hermes Group: This is Germany's second largest postal service provider. In its distribution center in Haldensleben (Germany) it operates full-service order fulfillment for one of Europe's largest online and catalogue fashion retailers. Hermes's order picking process is designed as follows. Each picker is equipped with a picking cart that carries up to four bins, each having a capacity of about 20 to 30 items. Pickers operate in fixed zones, each having a single drop-off point where new bins and the next pick list are retrieved and, finally, completed bins are handed over to the central conveyor system. Hence, each picker tour starts and ends at a central drop-off point, and for a given batch of (four) picking orders one seeks a tour through the respective zone of the warehouse, such that all items on the pick list can be retrieved, and the length of the tour is minimized.

Zalando: The online fashion retailer Zalando uses a different system at its distribution center in Erfurt (Germany). It applies static batching with a single drop-off point in each picking zone. Since the bins are relatively large and heavy when completely filled, picking carts are clumsy and inflexible. Hence, the pickers are much faster without the cart. Thus, a picker may park her cart, pick up to six items from close-by shelves, and carry them back to the cart. An optimization procedure for the Zalando case must therefore determine whether a walk between two successive visits of a picker tour should be executed with or without a cart. This decision has to consider the limited capacity of items a picker can carry on her arms and the varying walking speeds with and without a cart.

Based on these observations, we classify picking policies according to three characteristics: *i) static vs. dynamic batching*, *ii) single vs. multiple drop-off points*, and *iii) single vs. multiple travel modes (i.e., cartless subtours)*. Certain combinations of these characteristics reflect the real-world picking policies detailed above. In our computational studies, we aim at a full factorial design to evaluate the potential of each degree of freedom in designing picking policies. Hence, we add missing policies even if we have not yet observed them in practice. Table 1 shows the different picking policies with

increasing degrees of freedom from left to right. As can be seen, the Amazon (A) policy already covers most degrees of freedom, while the Hermes (H) and the Zalando (Z) policy remain at the bottom line with respect to degrees of freedom.

Table 1: Picking policies resulting from different attributes.

Policy	i (H)	ii (Z)	iii	iv	v	vi	vii (A)	viii
static (SB) / dynamic (DB) batching	SB	SB	SB	SB	DB	DB	DB	DB
single (SD) / multiple (MD) drop off points	SD	SD	MD	MD	SD	SD	MD	MD
no cartless subtours (NS) / cartless subtours (CS)	NS	CS	NS	CS	NS	CS	NS	CS

The table shows different picking policies with increasing degrees of freedom from left to right.

2.3 A generalized order picking problem

Given the multitude of picking policies described in Section 2.2, we now formally introduce the GOPP. In e-commerce, customer orders have different priorities and varying deadlines because some customers may take part in privileged delivery programs. However, considering these priorities directly leads to unrealistically long planning horizons of several hours that may even exceed the time needed to handle the orders known beforehand. In practice, the warehouse operator considers the priority of orders in line with the order batching and pick list sequencing at an upstream planning stage. The pick list sequence for a short planning horizon is then given to the picker. This sorted list sequence comprises the pick lists for the most urgent orders. The order waves that are not yet included in this planning horizon are more likely to be timely processed if the current set of pick lists is assembled as fast as possible.

Solution representation and notation

To represent a solution Π, we use information about the visited vertices $v \in \mathcal{V}$ which correspond to certain positions in the picking zone (Figure 2), such that a vertex is either a drop-off point ($v \in \mathcal{D}$), a crossing between an aisle and a cross-aisle ($v \in \mathcal{C}$), or a storage position ($v \in \mathcal{P}$) that contains varying amounts of different SKUs. At the operational level, an ordered set $\mathcal{L}$ of pick lists l denotes which position $v \in \mathcal{P}$ must be visited to pick up SKUs. Recall that the size and pick up position of a given SKU is decided at a preceding planning level to efficiently split customer orders on pick lists and avoid double picking by competing pickers. Hence we represent a single pick list $l = \{v_1, ..., v_n\}$ as a finite set of n storage positions that must be visited.

Besides, we use the set $\mathcal{L}_i^{\mathrm{a}}$ to track active pick lists at each position i of a route. Depending on the picking policy, a picker processes a single pick list or multiple pick lists in parallel such that $|\mathcal{L}_i^{\mathrm{a}}|$ is limited. Each pick list must be collected in a separate bin. Thus, we can measure the *picking cart capacity* κ, i.e., the number of possible parallel processed pick lists, by the number of bins. In practice, standardized bins are a prerequisite for fail-safe transport on picking carts and conveyors. Note that even for cases with differently sized bins, our assumption remains valid since the upstream planning stage avoids conflicting bins while defining the sequence of pick lists.

A picker is accompanied by a cart equipped with standardized bins to collect orders. For certain picking policies, the picker may temporarily leave her cart to pick a certain number of SKUs during a *cartless* *subtour*[†] .

A solution Π is an ordered set that defines the successive visits of the picker in $\mathcal{V}$. Each element of Π is a pair $\pi_i = (v_i, \mathcal{L}_i^{\mathrm{a}})$. The pair $\pi_0 = (v_0, \mathcal{L}_0^{\mathrm{a}})$ at the very first sequence position refers to the initial position of the picker, such that v_0 is fixed to the starting point of the picker. This ordered set defines a route through the picking zone such that a sequence of pick lists $\mathcal{L}$ is completed.

Objective function

The main factor that influences the total picking time at this operational stage is the picker walking

time since most other parts of the picker's time, e.g., the time to retrieve items from a shelf is fixed once the pick list is determined. Hence, the GOPP's objective is to minimize the total picker walking time, considering the picker's walking distance d_{v_{i-1},v_i} between consecutive vertices $v_{i-1}, v_i \in \mathcal{V}$ in Π

$$Z(\Pi) = \sum_{i \in \{1,\dots,|\Pi|\}} d_{v_{i-1},v_i}. \tag{1}$$

Constraints

In the following, we list the feasibility constraints for the GOPP. Some of these constraints differ depending on the type of batching, on the available travel modes, and on the number of drop-off points:

(a) The cart capacity κ is limited and $|\mathcal{L}_i^{\mathrm{a}}| \leq \kappa$ must hold for any pair π_i.

(b) Once a list starts being processed by a picker, it must be finished before a new pick list can be processed in the respective bin, i.e., each pick list has exactly one coherent *active interval*† .

(c) A pick list l cannot be completed before all visits of l are covered in its active interval.

(d) $\mathcal{L}$ is processed in its given order.

(e.s) For a static batching, the active intervals of pick lists either completely overlap or are completely disjoint.

(e.d) For a dynamic batching, (e.s) is relaxed. This allows handing over merely a subset of already completed active orders when stopping at a drop-off point.

(f) Bins can only be handed over at a drop-off point if the cart carrying them is also at the drop-off point, i.e., no drop-offs during cartless subtours are allowed.

(g) All pick lists must be completed, which is fulfilled when each pick list has exactly one active interval.

(h) If cartless subtours are allowed, only feasible subtours occur. A cartless subtour is feasible, if the capacity for carrying items of the picker is not exceeded and leaving and returning of hand carried items refer to the same parking position of the cart.

Among all feasible solutions fulfilling these constraints, the GOPP seeks a solution Π^* that minimizes the objective function (1) and hence contains active intervals of minimum length as stated in Proposition 1:

Proposition 1 *In an optimal solution, an active interval always remains as short as possible. It starts at the last drop-off point visit preceding the first storage position visit and ends at the first drop-off point succeeding the last storage position visit for each pick list.*

3 Methodology

In Appendix A we prove that the GOPP is NP-hard. However, a special variant of the GOPP can be solved in polynomial time. Specifically, for a setting with a single pick list, a single travel mode, and a single drop-off point, the rectilinear layout characterizing the GOPP (Figure 2) drastically limits the number of possible ways to traverse a sub-aisle (see Section 3.1). Leveraging this property, this version of GOPP can be solved in polynomial time. Accordingly, after summarizing the findings of previous work (Section 3.1), we first develop two skeletons for our algorithmic framework that solve a subproblem of the GOPP efficiently using a *layered graph algorithm* (LGA) (Section 3.2) or an integer linear program (ILP) (Section 3.3). We then derive a unified framework that allows using these skeletons to solve all variants of the GOPP based on branching and pruning rules (Section 3.4).

3.1 State of the art

A reduced version of the GOPP constitutes a special case of the rectilinear TSP which allows us to exploit some properties of this problem. Hence, for the sake of completeness, we summarize state-of-the-art developments in two fields: exact algorithms for picking problems, and recent enhancements of exact algorithms for the rectilinear TSP.

Exact algorithms in the field of order picking are still rare (cf. Section 1.1). To the best of our knowledge, only three algorithms exist, all of which are based on the work of Ratliff and Rosenthal (1983) who presented a first polynomial algorithm for a specified warehouse layout, namely a warehouse without interspersed cross-aisles. In a nutshell, the NP-hardness of the general TSP is removed since it can be proven that at most two arcs between adjacent vertices exist in a solution. This algorithm scales linearly with the number of sub-aisles since its complexity is significantly reduced by Proposition 2, which introduces the concept of *transitions* to denote a picker's movement through a sub-aisle(a_i, b_i).

Example 1 (Using transitions to model picker movements in sub-aisles) *Given a sub-aisle (a_i, b_i) that starts at a_i and ends at b_i, a transition denotes a possible way for a picker to travel through (a_i, b_i) in order to collect all items that she must pick in this sub-aisle. Figure 3 shows examples of such transitions for a sub-aisle (a_i, b_i).*

Proposition 2 [adapted from Ratliff and Rosenthal (1983)] *Let T denote a shortest path in G that visits every position with an item that must be picked, while G denotes a graph representation of a rectilinear warehouse. An optimal tour T in G contains only six different transitions of a sub-aisle (a_i, b_i).*

Figure 4 illustrates the *transitions* of Proposition 2 for a general representation of (a_i, b_i) with two (optional) stops in the sub-aisle. Note that, besides the transitions already detailed in Figure 3, these transitions contain an empty transition (I), because in an optimal solution, some sub-aisles which do not contain items that must be picked may not be visited. Summarizing, the complete set of transitions contains a non-traversed sub-aisle (I), and a complete traversal in one (II), or two (III) directions. Furthermore, a sub-aisle may be partially traversed with *slopes*† from one $(IV$,$V)$ or both (VI) directions. For transition (vi), the split between both slopes is always given by the longest distance between two stops in the same aisle if more than two stops are present in the sub-aisle. Although this split criterion may not be unique, it does not affect the global optimality of the solution since all split options have the same cost. For cross-aisles in which the picker does not pick up items, only transitions I–III, from here on referred to as standard transitions, are necessary.

Based on this rationale, Roodbergen and de Koster (2001) and Pansart et al. (2018) have proposed exact algorithms for warehouse layouts with three or more cross-aisles. However, these algorithms are limited in their applicability to the GOPP since multiple pick lists, parallel processing of pick lists, multiple drop-off points, and multiple travel modes are still not considered.

For rectilinear TSPs, Cambazard and Catusse (2018) proposed an $O(nh7^h)$ exact algorithm depending on the number of horizontal lines h and on the number of customers n. Proposition 3 lays the foundation for the algorithm's main rationale: for a TSP with a set of customer vertices $\mathcal{V}$, formally defined in a graph $G = (\mathcal{V}, \mathcal{A})$ with the vertex set $\mathcal{V}$ and the arc set $\mathcal{A}$, a *Hanan graph*† $H(\mathcal{U})$ with $\mathcal{U} \subseteq \mathcal{V}$ always contains a minimum length rectilinear Steiner tree for $\mathcal{V}$ if $H(\mathcal{U})$ is a *feasible Hanan graph*† for G.

Proposition 3 [adapted from Cambazard and Catusse (2018)] *We consider a graph $G = (\mathcal{V}, \mathcal{A})$ and a Hanan graph $H(\mathcal{U}) = (\mathcal{S}, \mathcal{E})$ with $\mathcal{U} \subseteq \mathcal{V}$. If $H(\mathcal{U})$ is a feasible Hanan graph for G, then a shortest path in $H(\mathcal{U})$ that contains all edges $e \in \mathcal{E}$ which cover vertices of $\mathcal{V}$, equals the optimal solution for the TSP in G.*

Note that an optimal *path*† in $H(\mathcal{U})$ denotes a *path subgraph*† T in G.

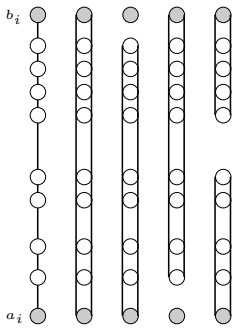


Figure 3: Examples of transitions for a sub-aisle.

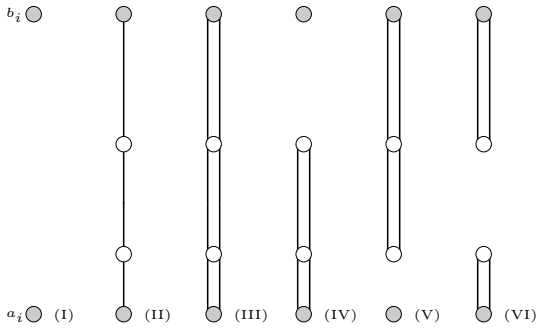


Figure 4: Sub-aisle transitions for optimal paths in G according to Ratliff and Rosenthal (1983).

3.2 A bidirectional layered graph algorithm for a single pick list

Figure 5 depicts a storage layout (a), and its corresponding graph representation (b) for a warehouse layout with a single drop-off point and one pick list. The drop-off point is the gray rectangle, and the shelves that must be visited are highlighted in orange. The set of intersections $\mathcal{C}$ yields a unique Hanan graph $H(\mathcal{C})$ whose edges span all mandatory storage positions $p \in \mathcal{P}$. By leveraging Proposition 3 we develop an algorithm based on the following core idea: instead of solving a rectilinear TSP on $G = (\mathcal{V}, \mathcal{A})$, we solve the Steiner variant of this problem on the Hanan graph $H(\mathcal{C}) = (\mathcal{C}, \mathcal{E})$ with $\mathcal{E}^{\mathrm{m}} \subset \mathcal{E}$ denoting the subset of edges that must be visited, from here on referred to as mandatory edges.

This problem can be solved with a standard LGA as discussed in Cambazard and Catusse (2018). In the following, we introduce a bidirectional layered graph algorithm which reduces the complexity of the state space and allows to solve problems more efficiently and for even larger instances. Note that while this algorithm was developed for the picking problem, it also appears to be the first available bidirectional algorithm for the rectilinear Steiner TSP.

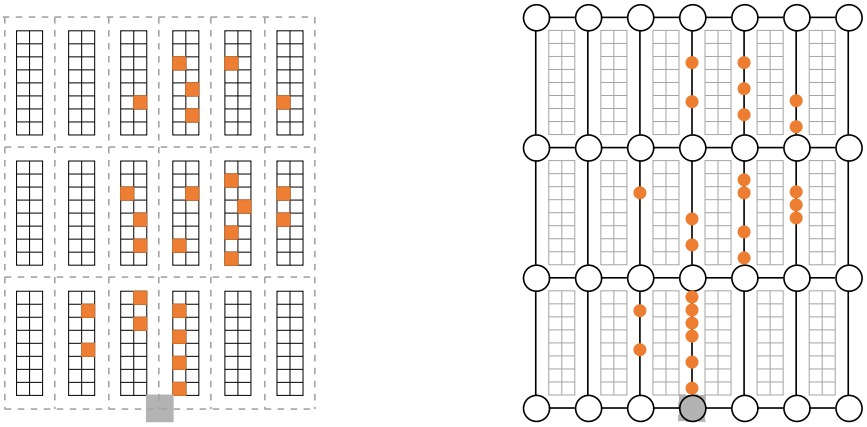
Proposition 4 [adapted from Cambazard and Catusse (2018)] *An optimal path subgraph T^* of G consists of the optimal partial path subgraph† to the left and to the right of a planar separator in G.*

Proposition 4 implies that we can split G and hence also $H(\mathcal{C})$ into two subgraphs which can be explored independently as long as we finally preserve degree parity and connectedness in the planar separator. To split our problem accordingly, we define a central planar separator S (Figure 6) that splits $H(\mathcal{C})$ into two subgraphs B and U. On both subgraphs, we define partial subgraphs, each uniquely defined by a horizontal i and a vertical j line in $H(\mathcal{C})$, counting horizontal lines from bottom to top and vertical lines from left to right (Figure 7). We define B_{ij} as a partial subgraph in B that covers all vertices lying in $[1, i] \times [1, j]$ or $[1, j-1] \times [i+1, h]$. The h vertices farthest to the right in B_{ij} define the right border R_{ij} of B_{ij}. Analogously, we define U_{ij} as the partial subgraph in U which covers all vertices that lie in $[j, v] \times [i, h]$ or $[j+1, v] \times [1, i-1]$; the h vertices farthest to the left in U_{ij} define the left border L_{ij} of U_{ij}.

Based on this, the main rationale of our algorithm is to propagate partial path subgraphs of $H(\mathcal{C})$ in B and U in order to complete these to a shortest feasible path to solve the Steiner TSP on $H(\mathcal{C})$. To propagate these partial path subgraphs, we use a transition set $\mathcal{T}_{v_i,v_j}$ which denotes all possible transitions $t \in \mathcal{T}_{v_i,v_j}$ for each edge (v_i, v_j).

Each partial path subgraph in B_{ij}/U_{ij} has a unique state $\alpha = \{\mathbf{r}_i, \mathbf{s}_i\}$ consisting of a vector $\mathbf{r}_i = (r_1, ..., r_h)$ which denotes the degree parity of each vertex in R_{ij}/L_{ij}, and of a vector $\mathbf{s}_i = (s_1, ..., s_h)$ which denotes the connected component to which each vertex in R_{ij}/L_{ij} belongs. Both the parity label r_i and the connected component label s_i refer to the vertex that lies on the i^{th} horizontal line of R_{ij}/L_{ij}.

We develop a bidirectional LGA that explores the partial subgraphs B_{ij}, U_{ij} in $H(\mathcal{C})$ by creating layered graphs $\mathcal{L}^{\mathrm{b}}$, $\mathcal{L}^{\mathrm{u}}$ with a dedicated layer for each B_{ij}/U_{ij}. On each layer we store feasible partial



(a) Physical layout. **(b) Graph representation.**

Figure 5: Example of a storage layout.

paths in B_{ij}/U_{ij} so that the optimal path can be identified by merging the last layers. Figure 9 provides the pseudo-code of our bidirectional LGA. We place a planar separator and identify it by the vertical index that denotes its position v^{s}, i.e., the aisle that forms the separator (line 5). We then propagate partial paths up to the forward and the backward propagation ends, coinciding with the separator (lines 6–7). Once R_{ij} equals L_{ij} in S (Figure 8), we obtain the optimal picking path by merging the states on the last layer of $\mathcal{L}^{\mathrm{b}}$ and $\mathcal{L}^{\mathrm{u}}$, thus identifying the lowest cost state that preserves feasibility in S (line 8). The superiority or inferiority of such a bidirectional algorithm compared to a monodirectional variant depends on the complexity of the merge procedure. As the merge procedure is highly technical we elaborate it in Appendix D to keep the paper concise.

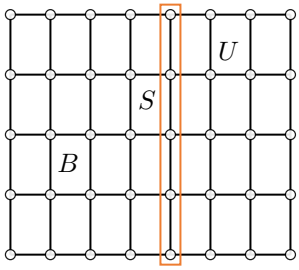


Figure 6: Example of S and its resulting subgraphs.

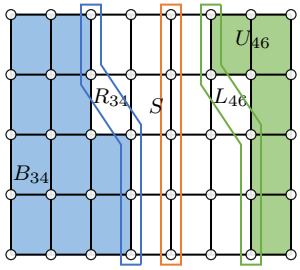


Figure 7: Example of partial subgraphs B_{43} and U_{64}.

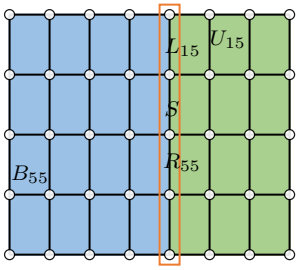


Figure 8: Fully explored graph with $R_{55} = S = L_{51}$.

Figure 10 provides the pseudo-code of our forward propagation mechanism. We identify each vertex v_{ij} by the horizontal (i) and the vertical (j) lines of $H(\mathcal{C})$ that cross in v_{ij}. For each B_{ij} we create a set $\mathcal{L}^{\mathrm{b}}_l$ that stores partial paths that can be realized in B_{ij}. Each path is associated with a state α having a cost $C^{\mathrm{b}}(\alpha, l)$. We explore B_{ij} starting at the bottom left corner (line 1), propagating vertical edges first (line 2) and horizontal edges iteratively (line 14). Each generated state α' results from a state α of the preceding layer and a transition t.

For each state, we check its feasibility (lines 6 and 17) first and remove the infeasible states. Then, α' is added to the set of states of the new layer (lines 11 and 12 and lines 22 and 23) or its cost value is updated if $\alpha' \in \mathcal{L}^{\mathrm{b}}_{l+1}$ already holds and its cost decreases. Note that partial paths that have an equal state are equivalent and hence, we only keep the path with the lowest cost. The backward propagation works analogously to the forward propagation but starts at the upper right of U_{ij}, propagating vertical edges first, and horizontal edges consecutively. Its pseudo code results by iterating $j \in \{v, ..., v^{\mathrm{s}}\}$ and $i \in \{h, ..., 1\}$, and considering $\mathcal{L}^{\mathrm{u}}$, C^{u}, respectively.

```
1: l ← 0; L_0^b ← ∅; L_0^b ← ∅
2: α_0 ← ((0, ..., 0), (−, ..., −))
3: L_0^b ← L_0^b ∪ {α_0}; L_0^u ← L_0^u ∪ {α_0};
4: C^b(α_0, 0); C^u(α_0, 0)
5: v^s ← ⌈v/2⌉
6: propagateFWD(l, C^b, L^b, v^s)
7: propagateBWD(l, C^u, L^u, v^s)
8: α* ← merge(C^b, L^b, C^u, L^u)
```

Figure 9: Pseudo-code of the bidirectional LGA.

```
1: for j ∈ {1, ..., v^s} do
2:     for i ∈ {1, ..., h} do
3:         for α ∈ L_l^b do
4:             for t ∈ T_{v_ij, v_{i+1,j}} do
5:                 α' ← α + t
6:                 if feasible(α', l + 1) then
7:                     if α' ∈ L_{l+1}^b then
8:                         if C^b(α', l + 1) > C^b(α, l) + c(t) then
9:                             C^b(α', l + 1) ← C^b(α, l) + c(t)
10:                    else
11:                        C^b(α', l + 1) ← C^b(α, l) + c(t)
12:                        L_{l+1}^b ← L_{l+1}^b ∪ {α'}
13:    if j ∉ S then
14:        for i ∈ {1, ..., h} do
15:            for t ∈ T_{v_ij, v_{i,j+1}} do
16:                α' ← α + t
17:                if feasible(α', l + 1) then
18:                    if α' ∈ L_{l+1}^b then
19:                        if C^b(α', l + 1) > C^b(α, l) + c(t) then
20:                            C^b(α', l + 1) ← C^b(α, l) + c(t)
21:                    else
22:                        C^b(α', l + 1) ← C^b(α, l) + c(t)
23:                        L_{l+1}^b ← L_{l+1}^b ∪ {α'}
24:    l ← l + 1
```

Figure 10: Pseudo-code of the forward propagation.

To apply this algorithm, we specify in the following *i*) transition sets to propagate partial path subgraphs, *ii*) a feasibility check that identifies partial path subgraphs that cannot become feasible early, and *iii*) a dominance criterion that allows to reject dominated partial path subgraphs early. Additionally, we present a time-efficient online preprocessing technique that improves our graph representation to speed up computational times significantly.

Transitions

To propagate partial path subgraphs for order picking problems, it is no longer sufficient to use only the three standard transitions, an empty, a single, or a double traversal on an edge, as pickers may traverse sub-aisles with slopes. Hence, we use the transitions proposed by Ratliff and Rosenthal (1983) as vertical transitions (cf. Figure 4). Doing so straightforwardly yields a transition set that is larger than needed. To derive a minimum transition set and speed up computational times, we separate all edges of set $\mathcal{E}$ of $H(\mathcal{C})$ into a set of mandatory edges $\mathcal{E}^{\mathrm{m}}$ and a set of optional edges $\mathcal{E}^{\mathrm{o}}$. Mandatory edges comprise edges that represent a mandatory sub-aisle while optional edges represent non-mandatory sub-aisles or cross-aisles such that $\mathcal{E} = \mathcal{E}^{\mathrm{m}} \cup \mathcal{E}^{\mathrm{o}}$. Since each edge in $\mathcal{E}^{\mathrm{m}}$ must be visited, transitions II–VI are sufficient to propagate mandatory edges. Non-mandatory edges will only be used as connecting edges such that we need only transitions I–III to propagate these.

Feasibility check

Our feasibility check is based on the degree r_i and on the connected component label s_i of each vertex in R_{ij}/L_{ij}. The degree of each vertex $r_i \in \{\mathrm{E}, \mathrm{O}, 0\}$ can be even and positive (E), odd (O), or zero (0). A partial path subgraph B_{ij}/U_{ij} may contain multiple partial paths that are not necessarily connected with each other (cf. Definition 7, Appendix B). To trace these, s_i denotes an integer label of the connected component of the partial path ending on the i^{th} horizontal line of R_{ij}/L_{ij}. If v_{ij} is not connected to R_{ij}/L_{ij}, s_i may remain empty. Note that after the merge operation, all s_i variables must be equal since the optimal path is not connected otherwise.

Given this information, we can check the feasibility of a transition as follows. A transition is infeasible if any of the following statements holds:

i) After applying a transition from vertex i to vertex j, i remains with an odd degree but is no longer in R_{ij}/L_{ij}.

ii) An empty transition is applied on a mandatory edge.
iii) The drop-off point vertex has only empty transitions.
iv) After applying a horizontal transition, a connected component is reduced without a merge.

To check the feasibility during the merge operation, a merge is feasible only if:

i) No vertex in S remains with an odd degree.
ii) All vertices in S have an equal connected component label.

Dominance criterion
To keep the number of propagated partial path subgraphs as small as possible, we withdraw each partial path subgraph that is equivalent to an already existing partial path subgraph and does not improve its cost. This criterion is based on the rationale that if two *equivalent partial path subgraphs*† T and T' in L_{ij} have equal cost, then it is sufficient to keep only one of them to derive an optimal path subgraph. We establish this equivalence in Proposition 5.

Proposition 5 *Two distinct partial path subgraphs T and T' in L_{ij} are equivalent if $\alpha_T = \alpha_{T'}$*

Improved graph representation
With the modifications described above, we obtain an LGA that is sufficient to solve a picker routing problem for a single pick list, a single drop-off point, and an arbitrary number of aisles and cross-aisles. In the following, we derive an improved graph representation that reduces the number of developed labels to enhance the algorithm's computational performance.

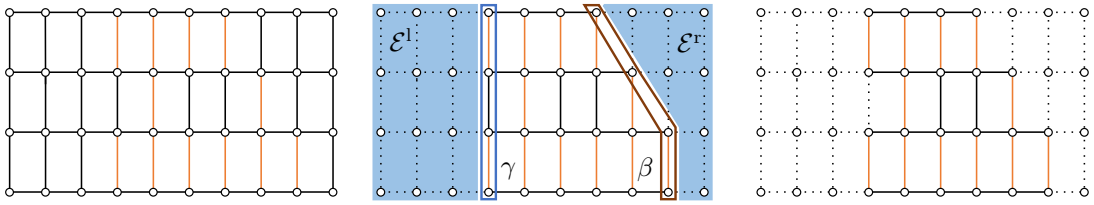
The number of labels increases with the number of edges that must be traversed. Hence, we aim to derive a maximal sparse Hanan graph H^{s} that preserves optimality but contains as few edges as possible. A minimum Manhattan network (MMN)† of $H(\mathcal{C})$ represents such a graph H^{s}. However, generating a MMN from $H(\mathcal{C})$ is NP-complete (Chin et al., 2011) and would add a non-polynomial component to our algorithm. Hence, in the following we provide an alternative reduction technique that improves the sparsity of $H(\mathcal{C})$.

For the following discussions we divide the set of edges into a set of sub-aisles $\mathcal{E}^{\mathrm{s}}$ and a set of subcross-aisles $\mathcal{E}^{\mathrm{c}}$. Recall that we index aisles from the left to the right and cross-aisles from bottom to top. Then, we refer to $\mathcal{E}^{\mathrm{s}}_i,\ i \in \{0, ..., v\}$ as all sub-aisles that belong to aisle i and $\mathcal{E}^{\mathrm{h}}_j,\ j \in \{0, ..., h-1\}$ denotes all sub-aisles in between the j^{th} and $(j+1)^{\mathrm{th}}$ cross-aisle.

We base the following discussion on the example shown in Figure 11. Figure 11a depicts a picking layout with mandatory sub-aisles (i.e., sub-aisles that must be traversed) highlighted in orange. To derive an approximation for the MMN, we identify a set of edges $\mathcal{E}^{\mathrm{l}} : \exists(i,j) \in \mathcal{E}^{\mathrm{s}}_0$ and a set of edges $\mathcal{E}^{\mathrm{r}} : \exists(i,j) \in \mathcal{E}^{\mathrm{s}}_v$ that can be removed from $\mathcal{E}$ to reduce $H(\mathcal{C})$ based on Proposition 6. Herein, we refer to the right border of $\mathcal{E}^{\mathrm{l}}$, i.e., the planar separator on the right of the subgraph that results out of $\mathcal{E}^{\mathrm{l}}$, as γ. Analogously, we define the left border of $\mathcal{E}^{\mathrm{r}}$ as β. We refer to such a *reduced Hanan graph* as $H' = (\mathcal{C}', \mathcal{E}')$ with a reduced edge set $\mathcal{E}'$ and a vertex set $\mathcal{C}'$ induced by $\mathcal{E}'$.

Proposition 6 *The reduced Hanan graph $H' = (\mathcal{C}', \mathcal{E}')$ with $\mathcal{E}' = \mathcal{E} \setminus \{\mathcal{E}^l \cup \mathcal{E}^r\}$ approximates the MMN of $H(\mathcal{C})$ and is sparser than $H(\mathcal{C})$, i.e., $|\mathcal{C}'| \leq |\mathcal{C}|,\ |\mathcal{E}'| \leq |\mathcal{E}|$, but less sparse than its MMN if β and γ preserve a concave shape; and both $\mathcal{E}^l$ and $\mathcal{E}^r$ do not contain mandatory sub-aisles.*

Figure 11b depicts $\mathcal{E}^{\mathrm{l}}$, $\mathcal{E}^{\mathrm{r}}$, γ, and β for our example. Figure 11c shows an infeasible reduced graph for which Proposition 6 does not hold. Based on Proposition 6, we can create H' in $O(|\mathcal{E}|)$ time, traversing the graph only once. For some special, synthetic, cases in which there exists *i*) a non-mandatory aisle, i.e., an aisle without mandatory sub-aisles, in between mandatory aisles and *ii*) a subset $\mathcal{E}^{\mathrm{h}}_j$ comprises no mandatory sub-aisle, we treat the sub-aisle at the intersection between the non-mandatory aisle and $\mathcal{E}^{\mathrm{h}}_j$ as *pseudo-mandatory*† to keep Proposition 6 valid.



(a) Example instance. **(b) Feasible reduced graph.** **(c) Infeasible reduced graph.**

Figure 11: Examples of feasible and infeasible upper bounds on the MMN of H.

3.3 A mixed integer program for a single picklist

Instead of designing an LGA to solve the order picking problem for a single depot and a single pick list, we can also develop a concise and efficient ILP to solve this problem. We define this ILP on a modified, directed graph $G' = (\mathcal{C}', \mathcal{A})$ which results from transforming the reduced graph H' into a directed graph. In addition to the set of all arcs $\mathcal{A}$, $\mathcal{A}^{\mathrm{m}}$ denotes all mandatory arcs, i.e., arcs corresponding to sub-aisles that must be traversed, and $\mathcal{A}^{\mathrm{d}\text{-}}, \mathcal{A}^{\mathrm{d}+}$ denote the subset of arcs leaving or entering the vertex that represents the drop-off point. Further, $\mathcal{A}_i$ denotes for each aisle i, the subset of cross-aisle arcs that are located between the i^{th} and the $(i+1)^{\mathrm{st}}$ aisle. We use the cut sets $\delta^+(a)$ and $\delta^-(a)$ to denote all predecessor and successor arcs of a, and a' to denote the inverse arc of a and define $\overline{\mathcal{A}}^{\mathrm{m}} \subseteq \mathcal{A}^{\mathrm{m}} : i < j \ \forall (i,j)$. We use binaries x_a to indicate whether an arc is traversed ($x_a = 1$) or not ($x_a = 0$) and binaries y_{at} to indicate whether an arc is sloped with a specific transition t ($y_{at} = 1$) or not ($y_{at} = 0$). Since the traversals already cover some of the original transitions, we use a limited transition set $\mathcal{T}' = \{IV, V, VI\}$ (see Figure 4) to specify each $t \in \mathcal{T}'$. The coefficients c_u and c_{ut} denote the cost for each traversal or slope. Further, we use binary z_{ab} to trace if in a tour, arc b is the direct successor of arc a. Binary o_a identifies the first arc of a tour, while binary d_a identifies the last arc of a tour.

With this notation as summarized in Table 2, our ILP is as follows:

Table 2: Notation used in the MIP formulation.

$\mathcal{A}$	set of all arcs	c_u	cost of traversing arc a
$\mathcal{A}^{\mathrm{m}}$	set of all mandatory arcs	c_{ut}	cost of sloping arc a with transition t
$\mathcal{A}^{\mathrm{d}\text{-}}$	set of all depot leaving arcs	a'	inverted arc of a
$\mathcal{A}^{\mathrm{d}+}$	set of all depot entering arcs		
$\mathcal{A}_i$	cross-aisle arcs between the i^{th} and $(i+1)^{\mathrm{st}}$ aisle	x_a	a is traversed ($x_a = 1$) or not ($x_a = 0$)
$\overline{\mathcal{A}}^{\mathrm{m}}$	mandatory arc subset ($\overline{\mathcal{A}}^{\mathrm{m}} \subseteq \mathcal{A}^{\mathrm{m}} : i < j \ \forall (i,j)$)	y_{at}	a is covered with slope t ($y_{at} = 1$) or not ($y_{at} = 0$)
$\mathcal{T}'$	limited transition set ($\mathcal{T}' = \{IV, V, VI\}$)	z_{ab}	b is traversed after a ($z_{ab} = 1$) or not ($z_{ab} = 0$)
$\delta^-(\mathcal{B})$	cut set yielding all successor arcs of $\mathcal{B} \subset \mathcal{A}$	o_a	u is the first arc of the tour ($o_a = 1$) or not ($o_a = 0$)
$\delta^-(a)$	cut set yielding all successor arcs of a	d_a	a is the last arc of the tour ($d_a = 1$) or not ($d_a = 0$)
$\delta^+(a)$	cut set yielding all predecessor arcs of a		

$$\text{minimize} \quad Z = \sum_{a \in \mathcal{A}} c_u x_a + \sum_{a \in \mathcal{A}^{\mathrm{m}}, t \in \mathcal{T}'} c_{ut} y_{at} \tag{2}$$

$$\text{subject to} \quad x_b \geq \sum_{a \in \delta^+(b)} z_{ab} \qquad b \in \mathcal{A} \tag{3}$$

$$x_a \geq \sum_{b \in \delta^-(a)} z_{ab} \qquad a \in \mathcal{A} \tag{4}$$

$$\sum_{b \in \delta^-(a)} z_{ab} \geq x_a \qquad a \in \mathcal{A} \setminus \mathcal{A}^{\mathrm{d+}} \tag{5}$$

$$\sum_{b \in \delta^-(a)} z_{ab} + d_a \geq x_a \qquad a \in \mathcal{A}^{\mathrm{d+}} \tag{6}$$

$$\sum_{a \in \delta^+(b)} z_{ab} \geq x_b \qquad b \in \mathcal{A} \setminus \mathcal{A}^{\mathrm{d-}} \tag{7}$$

$$\sum_{a \in \delta^+(b)} z_{ab} + o_a \geq x_b \qquad b \in \mathcal{A}^{\mathrm{d-}} \tag{8}$$

$$x_a + x_{a'} + \sum_{t \in \mathcal{T}'} y_{at} \qquad a \in \overline{\mathcal{A}}^{\mathrm{m}} \tag{9}$$

$$\sum_{a \in \mathcal{A}^{\mathrm{d-}}} o_a = 1 \tag{10}$$

$$x_a \geq o_a \qquad a \in \mathcal{A}^{\mathrm{d-}} \tag{11}$$

$$\sum_{a \in \mathcal{A}^{\mathrm{d+}}} d_a = 1 \tag{12}$$

$$x_a \geq d_a \qquad a \in \mathcal{A}^{\mathrm{d+}} \tag{13}$$

$$\sum_{t \in \{IV, VI\}} y_{at} \leq \sum_{b \in \delta^+(a)} x_b \qquad a \in \overline{\mathcal{A}}^{\mathrm{m}} \tag{14}$$

$$\sum_{t \in \{V, VI\}} y_{at} \leq \sum_{b \in \delta^-(a)} x_b \qquad a \in \overline{\mathcal{A}}^{\mathrm{m}} \tag{15}$$

$$\sum_{a \in \mathcal{A}_i} x_a \geq 2 \qquad i \in \{1, ..., n-1\} \tag{16}$$

$$\sum_{a \in \mathcal{B}} x_a \leq \sum_{\delta^-(\mathcal{B})} x_a |\mathcal{A}| \qquad \mathcal{B} \subseteq \mathcal{A} \setminus \{\mathcal{A}^{\mathrm{d+}} \cup \mathcal{A}^{\mathrm{d-}}\} \tag{17}$$

$$x_a, y_{at}, z_{ab}, o_a, d_a \in \{0, 1\} \qquad a, b \in \mathcal{A}. \tag{18}$$

Our objective minimizes the total cost, which results from traversing or sloping arcs. Constraints (3) and (4) link arc and sequence binaries, Constraints (5) and (6) ensure that each but the last traversed arc has a successor, and similarly Constraints (7) and (8) enforce the existence of a predecessor for all but the first traversed arc. Constraints (9) ensure that every sub-aisle that must be visited is traversed or sloped. Constraints (10) and (11) uniquely identify the first arc of a tour, while Constraints (12) and (13) uniquely identify the last arc of a tour. Constraints (14) and (15) force each slope to be connected to a traversal and impose that each subset of cross-aisles that remains in the reduced graph between two aisles is traversed at least twice. Constraints (17) ensure connectivity. Finally, Constraints (18) define the domains of the variables.

Since the number of Constraints (17) grows exponentially with the size of $\mathcal{A}$, we add these in a lazy fashion as feasibility cuts in the solver.

3.4 Extensions for the general GOPP

We now present a general framework that allows to integrate either the LGA algorithm of Section 3.2 or the ILP of Section 3.3 to solve any variant of the GOPP. We first discuss characteristics of and extensions necessary to integrate *cartless subtours*, a *dynamic batching policy*, and *multiple drop-off points* in Section 3.4.1. We then detail a general branching and pruning framework that serves as a paramount algorithm in Section 3.4.2.

3.4.1 Characteristics and lower bounds

To formalize the integration of dynamic batching and multiple drop-off points, we refer to a solution $\sigma = (\vartheta_1, ..., \vartheta_n)$ of the GOPP as an n-tuple of subproblem solutions ϑ_i. Each solution is associated with a cost $c(\sigma) = \sum c(\vartheta_i)$. A subproblem solution represents a single picking tour between two drop-off point visits and is a quadruple $\vartheta_i = (\mathcal{L}_i^{\mathrm{a}}, \mathbf{T}_i, d_i^-, d_i^+)$ consisting of a set of active picking list labels $\mathcal{L}_i^{\mathrm{a}} \subseteq \mathcal{L}$ which depends on the general set of order list labels $\mathcal{L}$ to be processed, a vector of transitions $\mathbf{T}_i$ that comprises an explicit transition $t_{ij} \in \mathcal{T}_{ij}$ for each edge, a starting drop-off point d_i^-, and a final drop-off point d_i^+. The cost for a subproblem solution results from the cost of its transitions.

Cartless subtours

Allowing for cartless subtours results in varying travel speeds in sub-aisles, which affect the cost of the transitions. We prove in Proposition 7 that the transitions discussed in Section 3.2 still suffice to calculate an optimal solution. Thus, our algorithm can be applied to problems with cartless subtours without changes.

Proposition 7 *If a picker can switch her travel mode from picking SKUs accompanied by her cart to picking a limited number of SKUs on foot, the transition set $\mathcal{T}$ is sufficient to derive an optimal solution.*

Therefore, no changes to our algorithm are necessary to handle cartless subtours. We only add a routine that identifies cartless subtours and the corresponding travel time reductions during postprocessing.

Dynamic batching

In the following, we first discuss the integration of dynamic batching still assuming that $d^+ = d^-$. Under a dynamic batching policy, a picker can pick multiple lists $l \in \mathcal{L}$ at once (as long as the carts bin capacity κ is not exceeded) and may return to the drop-off point once a pick list is finished to empty her bin. Accordingly, each subproblem denotes the possible transitions to form a picker tour between two drop-off point visits and $\mathbf{T}$ depends in each subproblem on the order lists that are active. Hence, the decision on which pick lists are active in parallel influences the number of consecutive subproblems in a subproblem sequence Θ. To exploit the characteristics of dynamic batching and reduce its combinatorial complexity, we prove Propositions 8 and 9.

Proposition 8 *Traversing a sub-aisle without collecting all items of at least one active pick list results in additional travel time.*

Proposition 9 *The number of picking tours, i.e., the number of subproblems, in the optimal solution does not exceed $m = \lceil |\mathcal{L}|/\kappa \rceil$.*

Proposition 9 implies that for the optimal solution σ^*, the number of subproblems always results to $|\sigma^*| = \lceil |\mathcal{L}|/\kappa \rceil$. Because order lists must be processed in chronological order (cf. Section 2.3), this further limits the permutations of active picklists.

Multiple drop-off points

In a setting with multiple drop-off points, vertices with odd degrees may arise in a subproblem ϑ_i with $d_i^+ \neq d_i^-$. We prove that these odd degrees are always limited to d_i^+ and d_i^-.

Proposition 10 *The optimal path subgraph T^* in ϑ_i with $d_i^+ \neq d_i^-$ comprises exactly two vertices with an odd degree which relate to d_i^+ and d_i^-.*

With Proposition 10, we extend the feasibility check of our LGA to a subproblem with $d_i^+ \neq d_i^-$ by enforcing an odd instead of an even degree for d_i^+ and d_i^-. Note that our ILP can also be directly used to solve such a problem setting by considering all depots in the respective arc sets.

Beyond this, we cannot achieve a complexity reduction for multiple drop-off points as this characteristic makes the GOPP NP-hard (Appendix A).

Lower bounds for a single subproblem

Considering multiple drop-off points requires to solve the GOPP using a branch-and-bound algorithm to identify the best drop-off configuration for a sequence of subproblems Θ. To enhance this algorithm in a branch-and-prune fashion, we develop two lower bounds Υ_1 and Υ_2 for a subproblem ϑ_i to prune branches early in the branch-and-bound tree.

To calculate Υ_1, we consider our original graph representation $G = (\mathcal{A}, \mathcal{V})$. We derive a minimal spanning tree on a reduced graph $G^c = (\mathcal{A}^c, \mathcal{V}^c)$, where $\mathcal{V}^c$ contains the depot and every pick-up position of ϑ_i, $\mathcal{A}^c = \{(i,j) \in \mathcal{A}, i \in \mathcal{V}^c, j \in \mathcal{V}^c\}$, and G^c is weighted with the Manhattan distance d_{ij} of each arc. A minimal spanning tree is given by $P^c \subseteq \mathcal{A}^c$ with a minimal total weight which yields Υ_1 such that

$$\Upsilon_1 = \sum_{(i,j)\in P^c} d_{ij}. \tag{19}$$

To calculate Υ_2, we consider the reduced Hanan graph $H' = (\mathcal{E}', \mathcal{C}')$. We derive a lower bound by summing up *i)* the cost-minimum transition for each mandatory edge, i.e., $d^*_{ij} = \min_{t\in\mathcal{T}^{ij}}\{d_{ijt}\}$, and *ii)* costs d_{III} for a double traversal of cross-aisles between each pair of aisles with n being the number of aisles remaining in H', such that

$$\Upsilon_2 = \sum_{(i,j)\in\mathcal{E}^{\mathrm{m}}} d^*_{ij} + d_{III}(n-1). \tag{20}$$

If cartless subtours are considered in ϑ_i, we treat all transitions as cartless to calculate Υ_2.

3.4.2 Branch-and-prune algorithm

To solve the GOPP in its most general fashion, one needs to decide on *i)* $\mathcal{L}_i^{\mathrm{a}*}$, i.e., which pick lists are processed in which subproblem, *ii)* which drop-off points d_i^{+*}, d_i^{-*} should be used in each subproblem ϑ_i, and on *iii)* $\mathbf{T}_i^*$, i.e., which transitions form the cost optimal picking tour. While we can solve *iii)* easily using either our LGA or our ILP as an oracle to determine the minimum cost and corresponding transitions, taking decisions *i)* and *ii)* requires a paramount algorithm which we develop in the following.

We refer to $\mathcal{L}^{\mathrm{a}}$ as an ordered set of $\mathcal{L}_i^{\mathrm{a}}$, denoting the set of active picking lists for each subproblem ϑ_i. Further, $\epsilon\mathcal{L}^{\mathrm{a}}$ denotes all feasible permutations of $\mathcal{L}^{\mathrm{a}}$ according to Section 3.4.1. To identify the right drop-off configuration for each subproblem, we use a branch-and-bound based approach. Herein, we create a search tree in which each level but the root node level depicts all drop-off point configurations for a subproblem. We represent a graph-theoretic rooted tree by using set theory: Let $\mathcal{N}$ be the set of node labels for the complete tree, with $\mathcal{N}_i \subset \mathcal{N}$ being a subset of node labels that are on the same layer of the tree, i.e., $|\mathcal{N}_0| = 1$ always holds and contains only the root node while $\mathcal{N}_1$ contains all siblings of the root node. As $\mathcal{N}_i$ is sufficient to identify the root node siblings unambiguously, $\mathcal{Z}_i$ denotes all siblings of node i. Each node i represents a subproblem, identified by a unique triple $(\mathcal{L}_i^{\mathrm{a}}, d_i^+, d_i^-)$. We define C^{ub} as a global upper bound as well as C_i^{ub} as an upper bound and C_i^{lb} as a lower bound in i, $\phi(i)$ calculates this lower bound by summing up costs for all subproblems on the path to n, solving the subproblem in n using either our LGA or our ILP, and considering lower bounds Υ_1 and Υ_2 for

the remaining unexplored subpaths. Let m denote the number of layers of our branch-and-bound tree. Note that on the last layer, $\phi(n)$ denotes the objective function value for the given path as no unsolved subproblems remain.

Example 2 (branch-and-bound tree) *Figure 12 shows an example of our branch-and-bound tree for a problem with two subproblems and drop-off points for a given $\mathcal{L}^a$. Clearly, $\mathcal{L}^a_1 = \mathcal{L}^a_2 = \mathcal{L}^a_3 = \mathcal{L}^a_4$ as all of these subproblems represent the same subproblem related to the active picking lists but differ in their drop-off point configuration.*

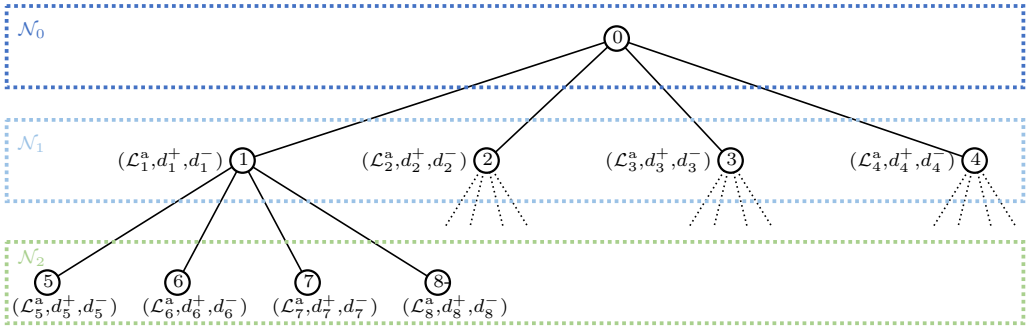


Figure 12: Branch-and-bound tree representation for a given $\mathcal{L}^a$ with $|\mathcal{L}^a| = 2$ and $|\mathcal{D}| = 2$.

Figures 13 and 14 detail the pseudo-code of our two-layered paramount algorithm which works as follows: On the first layer (Figure 13), we handle the dynamic batching decision. Given a small number of possible permutations (cf. Proposition 9) which is $|\epsilon\mathcal{L}^a| < 10$ even for application scenarios that are oversized, we rely on a simple enumeration for the dynamic batching decision and finally choose the $\mathcal{L}^a$ and the corresponding solution with the lowest cost. We call the second layer algorithm ($calculateBBtree()$) for each $\mathcal{L}^a$ to calculate its optimal cost. On the second layer (Figure 14), we use a branch-and-prune algorithm to determine the cost optimal drop-off point configuration for each subproblem. After building the branch-and-bound tree (line 1) based on $\mathcal{L}^a$ and potential drop-off points out of set $\mathcal{D}$, we calculate a first global upper bound by evaluating an initial path (line 2) for which all d^+_i and d^-_i equal the start drop-off point of the first subproblem. We add all nodes but the end node of the initial path and the nodes of the first layer $\mathcal{N}_1$ to the subset of active nodes and calculate their respective lower bounds (lines 2–5). We then proceed to explore the tree in a branch-and-prune fashion, always branching on n', the active node with the smallest lower bound (line 7). If n' is on the last layer of the tree (lines 12–14), we update the global upper bound as well as the local upper bounds of the respective path and prune the tree afterwards, herein, removing nodes from the set of active nodes if the new global upper bound is below their local lower bound. If n' is not on the last layer of the tree (lines 15–17), we calculate a lower bound for its siblings and add them to the set of active nodes. The algorithm finishes when the smallest lower bound matches the global upper bound (lines 8–10).

Combined with our LGA or our ILP, these algorithms provide an algorithmic framework to solve all variants of the GOPP. For further studies and application, we recommend to use this framework with our bidirectional LGA which yields the best computational performance (cf. Section 4.2.1), especially on large instances.

4 Computational study

This section details our computational study. We first present the scope of our experiments in Section 4.1, and we then discuss the results in Section 4.2.

```
for $\mathcal{L}^a \in \epsilon\mathcal{L}^a$ do
    $C(\mathcal{L}^a), \sigma(\mathcal{L}^a) \leftarrow calculateBBtree(\mathcal{L}^a, \mathcal{D})$
    if $C(\mathcal{L}^a) < C^*$ then
        $C^* \leftarrow C(\mathcal{L}^a)$
        $\sigma^* \leftarrow \sigma(\mathcal{L}^a)$
```

Figure 13: Pseudo-code for the first layer of the paramount algorithm.

```
$\mathcal{N}, \mathcal{Z}_n, \mathcal{N}_i \leftarrow createBBtree(\mathcal{L}^a, \mathcal{D})$
$C^{ub}, \mathcal{N}^a \leftarrow calculateInitialPath()$
$\mathcal{N}^a \leftarrow \mathcal{N}_1$
for $n \in \mathcal{N}^a$ do
    $C_n^{lb} \leftarrow \phi(n)$
while $\mathcal{N}^a \neq \emptyset$ do
    $n' \leftarrow n : \text{argmin}_{n \in \mathcal{N}^a} C_n^{lb}$
    if $C_{n'}^{lb} \geq C^{ub}$ then
        $C(\mathcal{L}^a) \leftarrow C^{ub}$
        return $C(\mathcal{L}^a), \sigma(\mathcal{L}^a)$
    $\mathcal{N}^a \leftarrow \mathcal{N}^a \setminus \{n'\}$
    if $n' \in \mathcal{N}_m$ then
        $C_n^{ub}, C_n^{lb} \leftarrow \phi(n)$
        $updateUB(C_n^{ub}, C^{ub}), prune(\mathcal{N}^a)$
    else
        for $n \in \mathcal{Z}_{n'}$ do
            $C_n^{lb} \leftarrow \phi(n), \mathcal{N}^a \leftarrow \mathcal{N}^a \cup \{n\}$
```

Figure 14: Pseudo-code for our branch-and-prune algorithm.

4.1 Design of experiments

This section embeds our experiments into real-world settings (4.1.1) before explaining them in detail. The goal of the experiments is threefold. First, we benchmark the basic component of our algorithm against the current state of the art (4.1.2). Second, since the current state of the art is not sufficient to evaluate real-time applications, we analyze the performance of our generic algorithmic framework in real-world environments with respect to different complexity drivers (4.1.3). Third, we study various warehouse layouts and picking policies based on three case studies (4.1.4).

4.1.1 Real-world settings

The three real-world applications discussed in Section 2.2 are similar with respect to the size and throughput of the warehouse. The sizes of these instances are simply staggering. Each comprises roughly 15,000,000 items of SKUs which is the total storage capacity of the Zalando warehouse. This translates into 400,000 items for a single picking zone. Focusing on spatial limitations, a picking zone has a width of 70 to 80 m, each aisle or cross-aisle has a width of 1.5 m, and a shelf has a width of 0.75 m. Each shelf stores about 300 items. Hence, 1,400 shelves are necessary to store 400,000 items. Choosing an average picking zone width of 75 m results in a layout with 25 aisles to cover 50 shelf rows, each row having 28 shelves. The width of a shelve is 2 m and hence we divide each shelve into two picking positions in our instance generation to derive a distance granularity of 1 m.

Whereas these characteristics are similar for all three warehouses, their layouts vary in terms of the number of cross-aisles and drop-off points: Zalando (Figure 15a) uses the most conventional layout with a single drop-off point (blue square) and three cross-aisles, at the beginning, in the middle, and at the end of the picking zone; Hermes (Figure 15b) works with a layout having a single drop-off point but uses five cross-aisles to increase the flexibility of the pickers; Amazon (Figure 15c), uses a layout with three cross-aisles but four additional drop-off points in the middle cross-aisle.

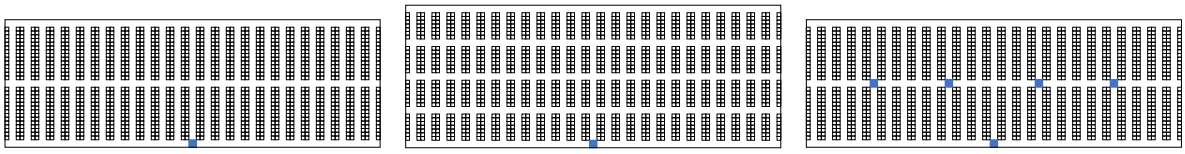
To allow for an unbiased analysis we need to work with equal pick list sequences. Hence, we cannot use real data from our three industrial cases. Instead, we randomly choose pick positions to create a sequence of pick lists which is a valid practice if scattered storage is applied (Weidinger and Boysen, 2018). We consider 35 picks per pick list, which correspond to a typical bin capacity for online retailers selling small to mid-size consumer goods. Since all layouts have the same number of aisles and shelves per aisle, this technique allows us to use the derived pick lists on each layout and hence to produce comparable unbiased results. Also, since we analyze scattered storages, this instance generation is sufficient to reflect real-world data. Further indication of the validity of this method is provided by the results of Weidinger (2018).

4.1.2 Experiment 1 – State of the art

Currently, no algorithm can solve all problem variants considered in this study. The most generic algorithm that was proposed in parallel to this work, by Pansart et al. (2018), focuses on a picker routing problem for a multiblock warehouse but is limited to a single pick list, a single drop-off point, a static batching, and does not consider cartless subtours. We compare the performance of both our monodirectional and our bidirectional LGA, and our ILP to the algorithm of Pansart et al. (2018). Since they also use dynamic programming we reimplemented their algorithm in our framework in order to allow a fair comparison on the same desktop computer. The benchmark from Theys et al. (2010) forms the basis for our comparison and comprises 1,080 instances which vary with respect to the number of aisles (5, 15, 60), the number of cross-aisles (3, 6, 11), the number of picks (15, 60, 240), a volume-based (V) or random (R) storage policy, and central as well as decentralized drop-off point positions. We compare the number of solvable instances and the average runtime to benchmark our algorithm.

4.1.3 Experiment 2 – Real-time applicability

Real-world environments bear several characteristics not covered in the benchmark of Experiment 1. Hence, we design a second benchmark to analyze the scalability of the computational performance of our algorithm for real-time applications. We analyze the impact of *i)* the number of cross-aisles and *ii)* the number of items per list and the size of the list sequence. Furthermore, we conduct these experiments with and without our preprocessing component to analyze its impact. In these experiments, we use at most $\kappa = 4$ parallel processed pick lists.



(a) Layout 1 (Zalando). **(b) Layout 2 (Hermes).** **(c) Layout 3 (Amazon).**

Figure 15: Picking zone layouts for three real-world cases, by increasing order of complexity.

4.1.4 Experiment 3 – Optimal picking policies

Through this experiment, we aim to derive insights into the optimal design of picking policies and the respective picking zone layouts. To this end, we perform a full factorial design with respect to the picking policy characteristics (Table 1) and the picking zone layouts from Section 4.1.1. To complete the experiment, we extend our sensitivity analysis as follows:

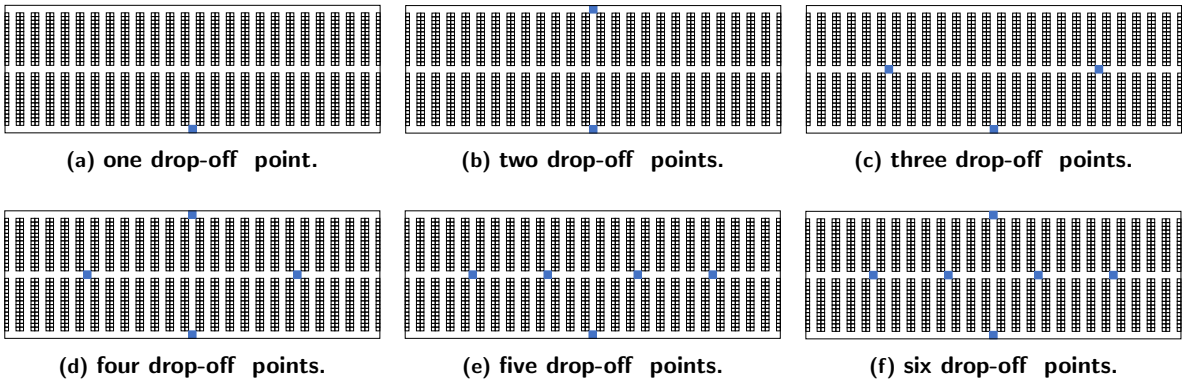
i) Taking the real-world layouts as an envelope to our study, we design picking zone layouts with one to six drop-off points and three and five cross-aisles. By enlarging the number of settings for additional drop-off point configurations beyond the configurations observed in practice, our study allows to derive more general insights into the benefit of picking zone layouts. Figure 16 shows an example of the drop-off point configurations for a layout with three cross-aisles.

ii) For all resulting layouts, we evaluate the impact of a) cartless subtours and b) dynamic batching. Note that we implicitly consider multiple drop-off points by each picking zone layout.

iii) The number of available bins, and therefore the number of parallel processed pick lists in a dynamic batching varies between two (Amazon, Zalando) and four (Hermes). Hence, we add an additional parameter scan over the number of available bins $\kappa \in \{2, 3, 4\}$ but we do not extend it to larger numbers of bins as the cart size is limited in practice.

For cartless subtours, we assume a picker walking speed of 5 km/h, whereas this speed goes down to 4 km/h if she is accompanied by her cart. We do not consider further variations in the picker walking speed as these values reflect physical properties which are hardly alterable.

Accordingly, this sensitivity analysis comprises 56 different settings that capture a multitude of picking policies and picking zone layouts that appear to be reasonable based on the three real-world cases. Note that, although extensive, these studies are not exhaustive since considering layouts with an even number of cross-aisles or further parameter studies may reveal additional benefits. However, we leave these studies for further research as they would require considerable space to allow for rigorous elaboration and discussion.



(a) one drop-off point. **(b) two drop-off points.** **(c) three drop-off points.**

(d) four drop-off points. **(e) five drop-off points.** **(f) six drop-off points.**

Figure 16: Example of the different drop-off point configurations for a layout with three cross-aisles.

4.2 Computational results

We now discuss our results, focusing on computational performance (4.2.1) and the impact of different picking policies (4.2.2).

4.2.1 Computational performance

We conducted all experiments on a standard desktop computer with an Intel Core I7 3.6GHz and 16 GB RAM. Our algorithm was implemented as a single thread code in C++. We solved the ILP using Gurobi 8.1.

Tables 3 and 4 compare the performance of our monodirectional (M-LGA) and bidirectional (B-LGA) algorithms, the algorithm of Pansart et al. (2018) (PCC), and our ILP in terms of *i)* the number of solved instances (Table 3) and *ii)* the computational times (Table 4). As can be seen, our LGAs perform much better than the algorithm of Pansart et al. (2018) in terms of both computational times and scalability on large-size instances. In particular, for large size instances, our LGAs show much better computational times and solve more instances. Comparing our monodirectional algorithm to our bidirectional algorithm, one can see that the bidirectional algorithm shows the best computational times and solves more instances with 11 cross-aisles then the monodirectional algorithm. Note that the computational times for the mono directional algorithm on instances with 11 cross-aisles and five aisles only appear to be lower since less instances are solved. Accordingly, the comparison is not complete as only solved instances are counted within the average computational times. Our ILP shows longer computational times than all problem specific algorithms but manages to solve a large subset of instances. Indeed, it solves nearly all instances with three and most instances with six and 11 cross-aisles. For the instances with three or six cross-aisles reported as not solved, it still finds an optimal solution but struggles to close the optimality gap within the time limit of an hour. Remarkably, the ILP solves the most instances with 11 cross-aisles out of all algorithms. While our bidirectional LGA solves 80 instances with 11 cross-aisles, the ILP solves 169 instances with 11 cross-aisles. For most

other instances with 11 cross-aisles, the ILP finds a solution but cannot prove its optimality within the given time limit. Only 21 instances remain completely unsolved.

The different performance of these solution approaches can be attributed to specific algorithmic components. Our LGA-based algorithms outperform that of Pansart et al. (2018) for three reasons: first, we use an elaborate online graph reduction which significantly reduces the number of states that must be explored. Notably, the time of this reduction routine is included within the computational times of Table 4. Second, we use improved transition sets which allow for a further reduction in the number of states that must be explored. Third, our feasibility check and equivalence criterion allow one to discard dominated or infeasible states early.

Table 3: Number of solved instances on the Theys et al. (2010) benchmarks.

	Volume based									Random								
Num. of. cross-aisles	3			6			11			3			6			11		
Num. of. aisles	5	15	60	5	15	60	5	15	60	5	15	60	5	15	60	5	15	60
B-LGA	60	60	60	60	60	60	31	23	14	60	60	60	60	60	60	10	2	-
M-LGA	60	60	60	60	60	60	24	16	13	60	60	60	60	60	60	3	-	-
PCC	60	60	60	60	60	60	-	-	-	60	60	60	60	60	60	-	-	-
ILP	60	60	58	60	59	34	54	39	6	60	60	43	60	55	8	42	28	-

The table shows the number of solved instances on the Theys et al. (2010) benchmark for the algorithm of Pansart et al. (2018) (PCC), our monodirectional LGA (M-LGA), our bidirectional LGA (B-LGA), and our ILP.

Table 4: Computational times on the Theys et al. (2010) benchmarks.

	Volume based									Random								
Num. of. cross-aisles	3			6			11			3			6			11		
Num. of. aisles	5	15	60	5	15	60	5	15	60	5	15	60	5	15	60	5	15	60
B-LGA	0.004	0.01	0.03	0.25	5.26	31.6	146	444	378	0.004	0.01	0.04	0.45	8.45	49.0	303	1779	-
M-LGA	0.006	0.02	0.04	1.14	7.49	36.8	80.3	639	532	0.006	0.02	0.05	2.04	11.98	56.5	263	-	-
PCC	0.007	0.02	0.10	3.67	40.8	490	-	-	-	0.006	0.02	0.07	2.96	22.5	345	-	-	-
ILP	0.030	0.95	40.6	0.38	41.4	0.42	419	429	422	0.050	3.30	124	18.0	76.7	0.65	635	728	-

The table shows the average computational times of the solved instances on the Theys et al. (2010) benchmark for the algorithm of Pansart et al. (2018) (PCC), our monodirectional LGA (M-LGA), our bidirectional LGA (B-LGA), and our ILP.

In total, the largest share of improvement is due to the graph reduction. Our bidirectional LGA outperforms its mono-directional counterpart as it can discard states earlier by exploring the subgraphs in two directions. While the computational complexity of our LGAs depends on the number of cross-aisles of an instance, the computational complexity of the ILP depends on the total number of arcs. Accordingly, it succeeds in solving a large number of instances with 11 cross-aisles and a small number of aisles but struggles on closing optimality gaps for instances with a large number of aisles, independent of the number of cross-aisles.

In real-world settings, the computational performance of our algorithm can be better or worse than in Experiment 1 due to a multitude of complexity drivers. In the following, we discuss this computational performance with respect to *i*) the number of cross-aisles, *ii*) the number of processed pick lists and the number of items on each list, and *iii*) the picking zone layout, and we show the impact of our preprocessing technique.

Table 5 shows computational results for a set of benchmark instances with a varying number of cross-aisles. We created 20 instances with a single depot and 50 aisles for each cross-aisle setting, generating pick positions as described in Experiment 3. We report the average computational time for each instance set. Further we analyze the ratio between the computational times without and with our graph reduction technique, from here on referred to as *speed-up factor*† . As can be seen,

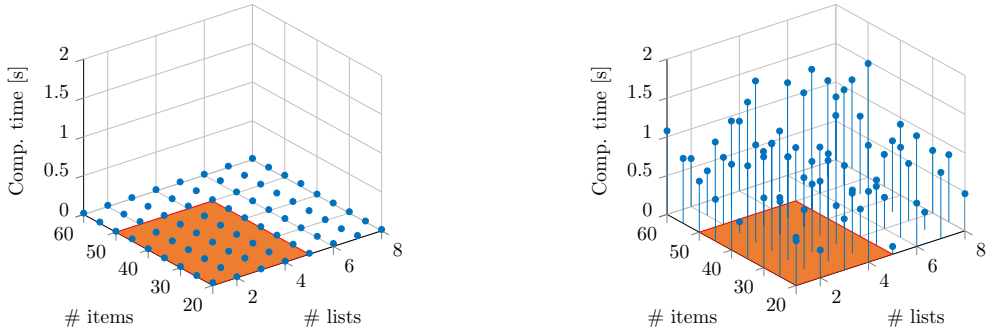
the overall complexity of the instances increases exponentially with the number of cross-aisles. The computational times remain stable for each instance set and settings up to five cross-aisles which are relevant in practice can be solved in less than a few seconds. Instances with seven cross-aisles already take a few minutes of computational time and instances with eight cross-aisles do no longer show stable computational times as some instances cannot be solved due to memory limitations. Without our graph reduction technique instances with seven cross-aisles already show unstable computational performance. The impact of the speed-up factor varies depending on the instance characteristics. The statistical quantities shown in Table 5 show that in approximately 75% of the cases our graph reduction technique reduces computational times at least 50%. In general, the improvement potential of this technique tends to be higher, the higher is the number of cross-aisles.

Table 5: Computational results for benchmark instance sets with 50 aisles.

Number of crossaisles	3	4	5	6	7	8
Average computational time [s]	0.03	0.13	1.72	17.4	551	-
Average speed-up factor	2.6	6.0	16.8	73.0	-	-
Maximum value speed-up factor	4.6	15.1	98.8	675.4	-	-
3^{rd} Quartile speed-up factor	3.2	7.6	16.3	45.2	-	-
Median speed-up factor	2.3	4.2	6.7	33.2	-	-
1^{st} Quartile speed-up factor	1.8	2.9	4.0	10.9	-	-
Minimum value speed-up factor	1.1	1.5	1.1	2.2	-	-

The table shows the average computational time and the statistical characteristics of the speed-up factor, i.e., the ratio between the computational time without and with preprocessing for benchmark sets of 20 instances for different numbers of cross-aisles.

Besides the number of aisles and cross-aisles, the number of consecutively processed pick lists and the number of items on each list influence the computational tractability of our algorithm. Furthermore, this tractability heavily depends on the chosen picking policy and on the number of drop-off points in a picking zone. The simplest real-world configuration comprises a static batching, a single drop-off point, and no cartless subtours on a layout with three cross-aisles and one drop-off point. The most complex real-world configuration comprises a dynamic batching (DS), multiple drop-off points (MD), and cartless subtours (CS) on a layout with three cross-aisles and five drop-off points (cf. Figure 15). Figure 17 shows the computational times for these two extreme configurations. Computational times remain stable below 0.1 second for the simple case. For the complex case, computational times remain below two seconds. These stable computational times in the complex case can be attributed to Proposition 9, which makes the algorithm rather insensitive to the number of consecutive pick lists. In practice, such algorithms are used in a receding horizon fashion within a 15-minute interval. This corresponds to a maximum number of four to five consecutive pick lists with up to 50 items. For these real-world configurations shown by the orange area in Figure 17, we achieve computational times below one second even for the worst scenario.



(a) Layout 1, (SB/SD/NS). **(b) Layout 3, (DB/MD/CS).**

Figure 17: Computational times dependent on the number of consecutive lists and items per list.

Concluding, our algorithm yields both new best known results for existing benchmarks and a generic framework that is amenable to a real-time implementation for a multitude of real-world configurations with respect to picking zone layouts and picking policies. In the following, we extend our experiments by using the bidirectional LGA as it provides the best and a robust performance on instance sizes relevant in practice.

4.2.2 Impact of different picking policies

To evaluate the performance of the different picking policies and picking zone layouts, we analyze the improvements in the total picking time. For each setting, we analyze a representative set of 50 instances with five consecutive pick lists of 30 items each. This corresponds to a 15-minute planning interval in a rolling horizon setting, which is often used in practice.

In the following, we report results based on average relative improvements which have been calculated by first comparing the results on a per instance basis and then computing an average value. Additionally, we applied one-tailed and two-tailed Wilcoxon signed-rank tests (Wilcoxon, 1945) to verify the tendency reflected by each average value.

We abbreviate each picking policy in the form $(\alpha/\beta/\gamma)$, with $\alpha \in \{\text{SB,DB}\}$ indicating whether batching is static (SB) or dynamic (DB), $\beta \in \{\text{SD,MD}\}$ indicating whether a single (SD) or multiple (MD) drop-off points are used, and $\gamma \in \{\text{NS,CS}\}$ indicating whether cartless subtours are allowed (CS) or not (NS).

To quantify the improvement potential of each picking zone layout and policy, Tables 6–8 show the average relative improvement in the total picking time for each setting and κ related to the average total picking time realized with a single depot and the most elementary picking policy (SB/SD/NS). Note that if a policy does not allow for multiple drop-off points, the comparison is skipped. As can be seen, the results reveal similar trends which vary only slightly in their amplitude depending on the number of consecutively processed pick lists. Here, a higher number of pick lists generally yields slightly lower improvement rates.

Table 6: Average policy and layout improvement potential with $\kappa = 2$.

	three cross-aisles						five cross-aisles						three vs. five cross-aisles					
# DPs	1	2	3	4	5	6	1	2	3	4	5	6	1	2	3	4	5	6
SB/SD/NS	0.00	-	-	-	-	-	0.00	-	-	-	-	-	-16.42	-	-	-	-	-
SB/SD/CS	-9.25	-	-	-	-	-	-7.99	-	-	-	-	-	-15.26	-	-	-	-	-
SB/MD/NS	0.00	-0.40	-1.16	-1.20	-1.44	-1.47	0.00	-0.81	-1.30	-1.36	-1.58	-1.61	-16.42	-16.77	-16.54	-16.56	-16.54	-16.54
SB/MD/CS	-9.25	-9.50	-9.69	-9.69	-9.88	-9.93	-7.99	-8.64	-8.83	-8.86	-8.91	-8.91	-15.26	-15.63	-15.63	-15.65	-15.51	-15.47
DB/SD/NS	0.00	-	-	-	-	-	0.00	-	-	-	-	-	-16.42	-	-	-	-	-
DB/SD/CS	-9.25	-	-	-	-	-	-7.99	-	-	-	-	-	-15.26	-	-	-	-	-
DB/MD/NS	0.00	-0.40	-1.16	-1.20	-1.44	-1.47	0.00	-0.81	-1.30	-1.36	-1.58	-1.61	-16.42	-16.77	-16.54	-16.56	-16.54	-16.54
DB/MD/CS	-9.25	-9.50	-9.69	-9.69	-9.88	-9.93	-7.99	-8.64	-8.83	-8.86	-8.91	-8.91	-15.26	-15.63	-15.63	-15.65	-15.51	-15.47

The table shows for $\kappa = 2$ the percentage deviation in total picking time for settings with three and five cross-aisles, taking the setting with policy (SB/SD/NS) and one depot as reference value. Further it shows the deviation between the three and the five cross-aisle settings, taking the respective three cross-aisle settings as reference value. Abbreviations hold as follows: SB - static batching, DB - dynamic batching, SD - single drop-off point, MD - multiple drop-off points, NS - no cartless subtours, CS - cartless subtours, #DPs - number of drop-off points.

As can be seen, using multiple drop-off points can yield improvements of up to 1.5%. However, these improvements' utility margin decreases for more than three drop-off points, and remain nearly equal between settings with five and six drop-off points. Allowing for cartless subtours yields improvements in between seven and nine percent, while applying a dynamic batching scheme does not yield any improvement in terms of total picking time for any setting. In the presence of cartless subtours, improvements that can be leveraged by multiple drop-off points show a slightly lower amplitude of up to 1.1%. Despite the innovative variations in picking policies, changing the picking zone layout from three to five cross-aisles yields the highest improvement for each setting, improving total picking time by up to 17%.

Focusing on the significant improvement gained by switching the picking zone layout from three to five cross-aisles, one should be aware of the fact that additional cross-aisles increase the flexibility of the

Table 7: Average policy and layout improvement potential with $\kappa = 3$.

	three cross-aisles						five cross-aisles						three vs. five cross-aisles					
# DPs	1	2	3	4	5	6	1	2	3	4	5	6	1	2	3	4	5	6
SB/SD/NS	0.00	-	-	-	-	-	0.00	-	-	-	-	-	-15.23	-	-	-	-	-
SB/SD/CS	-8.51	-	-	-	-	-	-8.25	-	-	-	-	-	-14.99	-	-	-	-	-
SB/MD/NS	0.00	-0.25	-0.89	-0.91	-1.14	-1.15	0.00	-0.51	-0.82	-0.85	-0.97	-0.99	-15.23	-15.45	-15.16	-15.18	-15.08	-15.09
SB/MD/CS	-8.51	-8.67	-9.31	-9.31	-9.40	-9.40	-8.25	-8.92	-8.95	-8.95	-8.85	-8.85	-14.99	-15.45	-14.88	-14.88	-14.69	-14.69
DB/SD/NS	0.00	-	-	-	-	-	0.00	-	-	-	-	-	-15.23	-	-	-	-	-
DB/SD/CS	-8.51	-	-	-	-	-	-8.25	-	-	-	-	-	-14.99	-	-	-	-	-
DB/MD/NS	0.00	-0.25	-0.89	-0.91	-1.14	-1.15	0.00	-0.51	-0.82	-0.85	-0.97	-0.99	-15.23	-15.45	-15.16	-15.18	-15.08	-15.09
DB/MD/CS	-8.51	-8.67	-9.31	-9.31	-9.40	-9.40	-8.25	-8.92	-8.95	-8.95	-8.85	-8.85	-14.99	-15.45	-14.88	-14.88	-14.69	-14.69

The table shows for $\kappa = 3$ the percentage deviation in total picking time for settings with three and five cross-aisles, taking the setting with policy (SB/SD/NS) and one depot as reference value. Further it shows the deviation between the three and the five cross-aisle settings, taking the respective three cross-aisle settings as reference value. Abbreviations hold as follows: SB - static batching, DB - dynamic batching, SD - single drop-off point, MD - multiple drop-off points, NS - no cartless subtours, CS - cartless subtours, #DPs - number of drop-off points.

Table 8: Average policy and layout improvement potential with $\kappa = 4$.

	three cross-aisles						five cross-aisles						three vs. five cross-aisles					
# DPs	1	2	3	4	5	6	1	2	3	4	5	6	1	2	3	4	5	6
SB/SD/NS	0.00	-	-	-	-	-	0.00	-	-	-	-	-	-13.14	-	-	-	-	-
SB/SD/CS	-7.58	-	-	-	-	-	-7.56	-	-	-	-	-	-13.11	-	-	-	-	-
SB/MD/NS	0.00	-0.33	-1.08	-1.12	-1.42	-1.44	0.00	-0.63	-1.00	-1.04	-1.21	-1.22	-13.14	-13.40	-13.08	-13.07	-12.96	-12.95
SB/MD/CS	-7.58	-7.85	-8.51	-8.53	-8.65	-8.65	-7.56	-8.31	-8.61	-8.61	-8.78	-8.78	-13.11	-13.57	-13.23	-13.21	-13.26	-13.26
DB/SD/NS	0.00	-	-	-	-	-	0.00	-	-	-	-	-	-13.14	-	-	-	-	-
DB/SD/CS	-7.58	-	-	-	-	-	-7.56	-	-	-	-	-	-13.11	-	-	-	-	-
DB/MD/NS	0.00	-0.33	-1.08	-1.12	-1.42	-1.44	0.00	-0.63	-1.00	-1.04	-1.21	-1.22	-13.14	-13.40	-13.08	-13.07	-12.96	-12.95
DB/MD/CS	-7.58	-7.85	-8.51	-8.53	-8.65	-8.65	-7.56	-8.31	-8.61	-8.61	-8.78	-8.78	-13.11	-13.57	-13.23	-13.21	-13.26	-13.26

The table shows for $\kappa = 4$ the percentage deviation in total picking time for settings with three and five cross-aisles, taking the setting with policy (SB/SD/NS) and one depot as reference value. Further it shows the deviation between the three and the five cross-aisle settings, taking the respective three cross-aisle settings as reference value. Abbreviations hold as follows: SB - static batching, DB - dynamic batching, SD - single drop-off point, MD - multiple drop-off points, NS - no cartless subtours, CS - cartless subtours, #DPs - number of drop-off points.

picker to shortcut into adjacent aisles, but add unused space to the warehouse and increase the lengths of the picking aisles. Hence, there exists a general trade-off between different design and operational objectives and we cannot conclude that more cross-aisles automatically increase picking performance. This trade-off has been explored in great detail for traditional (single-depot) settings in comprehensive computational studies, e.g., by Roodbergen and de Koster (2001) as well as Roodbergen et al. (2008).

Although Tables 6–8 allow us to analyze the improvement potential for different layouts and picking policies for a given κ, these results do not allow a quantification of the overall improvement potential depending on *i*) the picking policy, *ii*) the picking zone layout, and *iii*) an increase in κ.

Tables 9 and 10 detail the respective improvement potentials while increasing the number of parallel processed lists to $\kappa = 3$ (Table 9) or $\kappa = 4$ (Table 10) while taking the solution with $\kappa = 2$, as a baseline. As can be seen, by increasing the number of parallel processed lists to four, in the case of three cross-aisles an overall improvement of 23% can be reached, while the improvement in the case of five cross-aisles is with 20% slightly lower. Remarkably, 16%–17% of these improvements result by increasing κ from two to three, while increasing κ to four yields only an additional improvement of 4%–7%.

This can be explained by the relationship between κ and the total number of pick lists that must be processed. With $\kappa = 2$ and five pick lists, a picker must – independently of the batching policy – perform three consecutive tours to process all lists. Increasing the bin capacity to $\kappa = 3$ reduces the number of necessary consecutive tours to two, whereas increasing κ from three to four allows additional improvements but still requires two consecutive tours to process all pick lists. This effect may vary depending on the chosen setting, e.g., in a setting with $\kappa \in \{2, 3, 4\}$ and four consecutive pick lists, one would see the major share of the improvement when going from $\kappa = 3$ to $\kappa = 4$ as this would allow to reduce the number of consecutive tours from two to one.

Table 9: Average policy and layout improvement potential between $\kappa = 2$ and $\kappa = 3$.

	three cross-aisles						five cross-aisles					
# DPs	1	2	3	4	5	6	1	2	3	4	5	6
SB/SD/NS	-17.09	-	-	-	-	-	-15.92	-	-	-	-	-
SB/SD/CS	-16.41	-	-	-	-	-	-16.16	-	-	-	-	-
SB/MD/NS	-17.09	-16.96	-16.86	-16.85	-16.83	-16.82	-15.92	-15.66	-15.51	-15.49	-15.40	-15.39
SB/MD/CS	-16.41	-16.33	-16.73	-16.73	-16.64	-16.59	-16.16	-16.18	-16.03	-16.00	-15.87	-15.87
DB/SD/NS	-17.09	-	-	-	-	-	-15.92	-	-	-	-	-
DB/SD/CS	-16.41	-	-	-	-	-	-16.16	-	-	-	-	-
DB/MD/NS	-17.09	-16.96	-16.86	-16.85	-16.83	-16.82	-15.92	-15.66	-15.51	-15.49	-15.40	-15.39
DB/MD/CS	-16.41	-16.33	-16.73	-16.73	-16.64	-16.59	-16.16	-16.18	-16.03	-16.00	-15.87	-15.87

The table shows the improvement potential for each setting switching from $\kappa = 2$ to $\kappa = 3$. Abbreviations hold as follows: SB - static batching, DB - dynamic batching, SD - single drop-off point, MD - multiple drop-off points, NS - no cartless subtours, CS - cartless subtours, #DPs - number of drop-off points.

Table 10: Average policy and layout improvement potential between $\kappa = 2$ and $\kappa = 4$.

	three cross-aisles						five cross-aisles					
# DPs	1	2	3	4	5	6	1	2	3	4	5	6
SB/SD/NS	-23.74	-	-	-	-	-	-20.74	-	-	-	-	-
SB/SD/CS	-22.34	-	-	-	-	-	-20.37	-	-	-	-	-
SB/MD/NS	-23.74	-23.69	-23.68	-23.68	-23.72	-23.72	-20.74	-20.59	-20.50	-20.48	-20.45	-20.43
SB/MD/CS	-22.34	-22.35	-22.74	-22.75	-22.69	-22.65	-20.37	-20.46	-20.55	-20.52	-20.63	-20.63
DB/SD/NS	-23.74	-	-	-	-	-	-20.74	-	-	-	-	-
DB/SD/CS	-22.34	-	-	-	-	-	-20.37	-	-	-	-	-
DB/MD/NS	-23.74	-23.69	-23.68	-23.68	-23.72	-23.72	-20.74	-20.59	-20.50	-20.48	-20.45	-20.43
DB/MD/CS	-22.34	-22.35	-22.74	-22.75	-22.69	-22.65	-20.37	-20.46	-20.55	-20.52	-20.63	-20.63

The table shows the improvement potential for each setting switching from $\kappa = 2$ to $\kappa = 4$. Abbreviations hold as follows: SB - static batching, DB - dynamic batching, SD - single drop-off point, MD - multiple drop-off points, NS - no cartless subtours, CS - cartless subtours, #DPs - number of drop-off points.

Concluding, our results show that choosing the right picking policy can result in savings in total picker walking time of more than 20%, which provides potential to increase overall flexibility or throughput. However, the tasks pickers perform besides walking and multiple other objectives must also be considered. For instance, increasing the number of cross-aisles requires additional space and may generate problems at a strategic level. Further, dynamic batching can be used to increase flexibility by dropping finished lists early, but this benefit cannot be quantified by analyzing the total picker walking time.

5 Conclusions and managerial insights

With the rapid growth in e-commerce, picker routing and its associated policies have become crucial determinants of profitability and competitiveness in warehouse operations. We have presented the first generic and exact algorithmic framework for the GOPP, which constitutes a new state of the art for several known problem variants and for the generic case. Our algorithm is scalable and amenable to real-time applications since it requires computational times of less than two seconds for realistic warehouse layouts and picking policies. Using this algorithm we have analyzed the impact of several picking policies and warehouse layouts on the overall performance of a picker.

Although not exhaustive in all design aspects of picking zones, our results allow the derivation of several meaningful managerial insights, identifying trends that may open the field for further research. We have quantified the benefit of each picking policy and layout modification. We found that more flexible picking policies, the right bin capacity on the cart, as well as layouts with a higher number of cross-aisles, help increase the efficiency of the picking process significantly, by up to 23%. Our main insights can be summarized in the following key messages.

The improvement potential of both layout and policy modifications depends on the ratio between the number of in-parallel and consecutively processed pick lists. Our experiments demonstrate that the overall improvement potential heavily depends on the ratio between the number of pick lists executed in parallel and those consecutively processed. In general, the improvement potential decreases as the number of parallel processed pick lists becomes closer to the number of consecutive processed pick lists.

Modifying the number of cross-aisles may offset the improvement potential of modified picking policies. Independently of the number of in-parallel processed pick lists, a picking zone layout modification, in which the number of cross-aisles increases from three to five, always yields a better improvement than that observed for any change of picking policies in a layout with three cross-aisles.

Two of the picking policies yield significant improvements. Two of the picking policies have been shown to yield an improvement potential, independently of the number of cross-aisles and of the number of pick lists processed in parallel. Indeed, allowing for cartless subtours yields the highest improvement (7%–9%), followed by allowing for multiple drop-off points (up to 1.5%). Allowing for both cartless subtours and multiple drop-off points yields an additional improvement of about 1%.

Multiple drop-off points show a decreasing utility margin. Our experiments showed that the improvement gained by additional drop-off points decreases for high numbers of drop-off points. While increasing the number of drop-off points up to three still yields significant improvements, these tend to become marginal afterwards.

The improvement potential leveraged by flexible picking policies is higher for layouts with three cross-aisles. Our results show that in general, enabling flexible picking mechanisms yields a higher improvement for picking zone layouts with three cross-aisles, because a higher flexibility gain can then be leveraged. However, the improvement potential of different policies becomes more similar between layouts with three and five cross-aisles when the number of parallel processed lists becomes higher. This is the case because having more in-parallel processed lists also increases flexibility, independently of the picking zone layout.

Dynamic Batching does not affect the total picking time. Our results show that dynamic batching does not affect the total picking time. However, dynamic batching may yield other advantages, e.g., earlier dropping of completed lists, which plays an important role for same day or two-hour deliveries.

Two final comments in relation with these insights are in order. First, the impact of cartless subtours may be significantly higher or lower, based on the ratio between a picker's speed with and without a cart. The picker speeds used in this work are often observed in practice, which means that our results provide a good baseline, but our results should be reexamined if achievable speeds differ significantly. Second, although a dynamic batching policy and multiple drop-off points yield no improvements with respect to the total picking time, these practices generate benefits not analyzed in this work. If in addition to minimizing the total picking time one imposes the requirement that pick lists must be dropped as early as possible, dynamic batching and multiple drop-off points may offer additional flexibility not reflected in the total picking time.

A NP-hardness proof for the GOPP with multiple pick lists and multiple drop-off points

Here, we prove that the GOPP as defined in Section 2.3 (i.e., with multiple pick lists to be assembled according to a given processing sequence, a cart with a capacity of κ orders, and multiple drop-off points) is strongly NP-hard even if each pick list demands just a single item. Note that a straightforward transformation from the TSP is not available for our problem, because picker routing in parallel aisle

warehouses is solvable in polynomial time (Ratliff and Rosenthal, 1983). Instead, our transformation is from the sorting buffer problem (dubbed SBP), which is NP-hard in the strong sense (Asahiro et al., 2012). The SBP in its feasibility version is defined as follows.

> Sorting Buffer Problem: Given a sequence σ of n items, each item i is to be colored in color $c(i)$ by a server, which has a random access input buffer for up to k parallel items. According to σ, items are moved into the buffer. At any step, a color c is selected and all items in the buffer of color c are removed from the buffer and processed by the server. Freed buffer space is filled by succeeding items of σ; if some of these new items also demand color c, they are instantaneously removed until no item in the buffer is of color c. Then, a new color is selected and the process repeats, until no items remain in σ. Given no initial color, we have to decide whether a processing plan with at most C color changes exists.

We transform an instance I of SBP into an instance I' of our GOPP in polynomial time as follows. For each color c of I, we introduce an aisle within a rectilinear warehouse layout. Each item i with color $c(i)$ of SBR corresponds to a pick list demanding just a single item to be retrieved for a picking position located in the middle of the aisle corresponding to the respective color. Thus, if we have four items of color white in SBP, we introduce four pick lists each demanding an item stored at a picking positions in the middle of the 'white aisle' of GOPP. Furthermore, in the middle of each aisle we have a drop-off point. Note that, for a matter of convenience, we assume that all items stored in the same aisle and each aisle's drop-off point are located in the middle of the aisle and have zero distance among each other. This proof can easily be extended by (small) distances among an aisle's picking positions and drop-off point. Additionally, we have the picking cart of GOPP, which resembles the input buffer of SBP. The capacity of the cart equals that of the input buffer, so that we have $\kappa = k$. The sequence σ of SBP equals the GOPP sequence of pick lists. The length of each aisle is 2λ, so that the distance towards the middle of each aisle, where the picking positions and the drop-off points are located is λ. The distance all along the front cross-aisle in order to reach the parallel picking aisles is β. We assume $\lambda > C \cdot \beta$. Initially, the picker is located in the cross aisle and the question we ask is whether there exists a solution to I' with a total picking distance $Z < 2\lambda \cdot C$. Our transformation scheme is depicted in Figure 18.

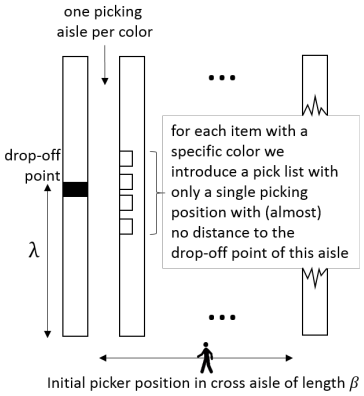


Figure 18: Transformation scheme of complexity proof.

If I is a yes-instance, visiting the aisles within GOPP in the same sequence as switching colors within SBP leads to a solution to I' with $Z < 2\lambda \cdot C$. When visiting an aisle with the picking cart all pick lists with their items stored in this aisle can be completed and handed over at the drop-off point. Since we have no distance among the picking positions of this aisle and the drop-off point, all succeeding pick lists put onto the cart according to given sequence σ and having their items stored in the current aisle can also be processed without causing additional picker walking. Thus, each aisle visit causes a distance of 2λ to walk back and forth the middle of the aisle, except of the last aisle

where picking stops after handing over the last pick lists. Since even completely passing the front cross aisle (of length β) C times does not exceed λ, the resulting solution to I' does not exceed $Z < 2\lambda \cdot C$.

On the other hand, any solution to I' of GOPP with $Z < 2\lambda \cdot C$ is also a feasible solution to I of SBP. Switching colors in the same sequence as changing aisles inevitably leads to C color changes, which completes our complexity proof.

B Definitions

This appendix lists all definitions in alphabetical order, followed by the section number where they are first mentioned.

Definition 1 (active interval) *[§2.3, p.8] Given a solution* Π*, a picker starts a not yet processed pick list* l *at a drop-off point preceding the first storage position visit related to* l*. Analogously, she hands over a completed pick list* l *at a drop-off point succeeding the last storage position visit related to* l*. During the time between these two drop-off point visits, we refer to an order as active and to the complete time span during which an order is active as its active interval.*

Definition 2 (cartless subtour) *[§2.3, p.7] A cartless subtour of the picker starts at the* i^{th} *stop of the tour where the picker leaves her cart and ends at stop* k *where she picks up her cart again at the same position such that* $v_i = v_k$*. In addition, a cartless subtour is limited to a certain number of items* K*, i.e., the maximum number of items a picker can carry without cart support. A picker may start such a subtour not only at the end of a sub-aisle, but at any point in it.*

Definition 3 (equivalent partial path subgraphs) *[§3.2, p.17] Let* T *and* T' *be two distinct partial path subgraphs in* L_{ij}*. We call* T *and* T' *equivalent if their optimal (shortest feasible) partial paths in* F *and* F' *are equal.*

Definition 4 (feasible Hanan graph) *[§3.1, p.9] Given a graph* $G = (\mathcal{V}, \mathcal{A})$ *and an arbitrary set of vertices* $\mathcal{U}$*, we call a Hanan graph* $H(\mathcal{U}) = (\mathcal{S}, \mathcal{E})$ *a feasible Hanan graph of* G *if every vertex* $v \in \mathcal{V}$ *is located on an edge of* $\mathcal{E}$*.*

Definition 5 (Hanan graph) *[§3.1, p.9] Let* $\mathcal{U}$ *be a finite set of vertices in a plane. We first construct a grid, made up of vertical and horizontal lines through each point of* $\mathcal{U}$*, commonly known as a Hanan grid (Hanan, 1966). The resulting intersections define a second set of vertices* $\mathcal{S}$ *which includes* $\mathcal{U}$*. The Hanan graph* $H(\mathcal{U}) = (\mathcal{S}, \mathcal{E})$ *formally defines this grid and consists of the vertex set* $\mathcal{S}$ *and an edge set* $\mathcal{E}$*, which contains all edges linking two consecutive vertices of* $\mathcal{S}$ *in the horizontal or vertical direction.*

Definition 6 (Minimum Manhattan Network) *[§3.2, p.13] Consider a set* $\mathcal{N}$ *of* n *points in* $\mathbb{R}^2$*. Then, a graph* G *forms a Manhattan network for* $\mathcal{N}$ *if* G *contains a rectilinear path for each pair* $n_1, n_2 \in \mathcal{N}$ *with a length equal to the Manhattan distance between* n_1 *and* n_2*. A minimum Manhattan network is a Manhattan network with minimum length, i.e., a Manhattan network having the minimum total length for its line segments.*

Definition 7 (partial path subgraph) *[§3.1, p.10] Let* $T = (\mathcal{V}^T, \mathcal{A}^T)$ *be a subgraph of* $L = (\mathcal{V}^L, \mathcal{A}^L)$ *which is a subgraph of* $G = (\mathcal{V}^G, \mathcal{A}^G)$*. Then,* T *is an* L *partial path subgraph of* G *if there exists at least one* $F = (\mathcal{V}^L, \mathcal{A}^L)$*,* $\mathcal{V}^L \subset \mathcal{V}^G \setminus \mathcal{V}^L$*,* $\mathcal{A}^L \subset \mathcal{A}^G \setminus \mathcal{A}^L$ *so that* T *and* F *form a path subgraph of* G*.*

Definition 8 (path) *[§3.1, p.10] A path* $p = (\sigma_0, \sigma_1, ..., \sigma_n)$ *in a Hanan graph* $H = (\mathcal{S}, \mathcal{E})$ *is given by a sequence of triples* $\sigma_i = (v_j, v_k, t)$*, where* $(v_j, v_k) \in \mathcal{E}$*, and* $t \in \mathcal{T}_{v_j, v_k}$ *is a transition.*

Definition 9 (path subgraph) *[§3.1, p.10] A subgraph* $T = (\mathcal{V}^T, \mathcal{A}^T)$ *of* G *is a path subgraph if it is connected and its degree* $\deg v > 0$*, and* $\deg v \mod 2 = 0$ *for* $v \in \mathcal{V}^T$*.*

Definition 10 (pseudo-mandatory) *[§3.2, p.13] We refer to a sub-aisle as pseudo-mandatory if it is not a mandatory sub-aisle, but is necessary to be kept in H' to preserve the connectivity of H'. Figure 19 shows an example of a graph with a pseudo-mandatory sub-aisle.*

Definition 11 (slope) *[§3.1, p.9] A slope denotes a (partial) traversal of a sub-aisle which starts and ends at the same crossaisle, i.e., the picker exits the sub-aisle right where he enters it.*

Definition 12 (speed-up factor) *[§4.2.1, p.22] The speed up factor denotes the ratio between a computational time of an algorithm achieved without applying our graph reduction technique divided by the computational time of the same algorithm on the same instance achieved with our graph reduction technique.*

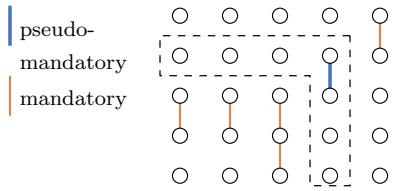


Figure 19: Example of a pseudo-mandatory sub-aisle.

C Proofs

Proof of Proposition 1. [§2.3 p.8] Starting (finishing) orders at a drop-off point earlier (later) than required cannot improve the solution value because in this case the cart capacity is blocked longer than necessary. □

Proof of Proposition 2. [§3.1 p.9] See Ratliff and Rosenthal (1983). □

Proof of Proposition 3. [§3.1 p.9] Since $H(\mathcal{U})$ is a feasible Hanan graph of G, all mandatory visits of the TSP lie on the edges $\mathcal{E}$ of $H(\mathcal{U})$. Further, $\mathcal{T}$ includes all feasible transitions that may arise in an optimal solution of both problems, and a Manhattan metric holds in both cases. Hence, the cost $c(p)$ of a path p is equal to that of an optimal path p^* in $H(\mathcal{U})$ and of an optimal path $\hat{p}$ in G, such that $c(p^*) = c(\hat{p})$ (Ratliff and Rosenthal, 1983). □

Proof of Proposition 4. [§3.2 p.10] See Cambazard and Catusse (2018). □

Proof of Proposition 5. [§3.2 p.12] To prove that T and T' are equivalent, we must show that *i)* T and T' allow for the same set of possible transitions in F resp. F', and *ii)* there exists an equivalent partial path in F and F' that forms a connected path with T. Let $f(v_i, v_j, r_i)$ be the function that denotes all feasible transitions for (v_i, v_j). Then, $f(v_i, v_j, r_i) = f(v_k, v_j, r_k)$ if $r_i = r_k$ holds according to Definition 9 and feasibility condition *i)*. This holds recursively in F if $r_i = r'_i$ for all $r_i \in R_{ij},\ r'_i \in R'_{ij}$. Given an equal basic set of transitions in F and F', an equivalent connected path in F and F' can be built if $s_i = s'_i$ for all $s_i \in R_{ij},\ s'_i \in R'_{ij}$. This concludes the proof since $\alpha_i = \alpha'_i \Leftrightarrow s_i = s'_i,\ r_i = r'_i$ for all $s_i, r_i \in R_{ij},\ s'_i, r'_i \in R'_{ij}$. □

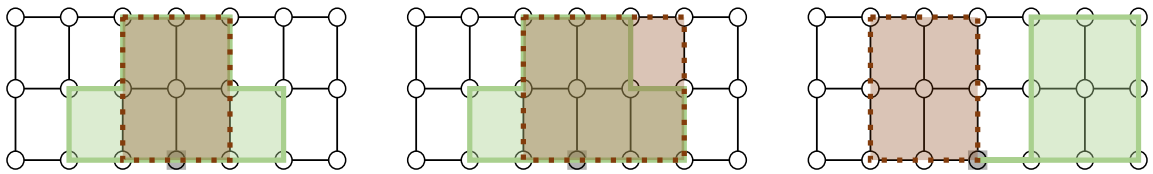
Proof of Proposition 6. [§3.2 p.13] We prove that the length of a shortest path for any pair of $(c_1, c_2),\ c_1, c_2 \in \mathcal{C}$ is the same in H' and H. This is the case if *i)* all active sub-aisles on an aisle and *ii)* H' remain connected, i.e., these sub-aisles are connected by its neighboring sub-aisles and enclosed sub-cross-aisles. Conditions *i)* and *ii)* only hold if the left and right border of H' are convex, which implies concave borders for $\mathcal{E}^{\mathrm{l}}$ and $\mathcal{E}^{\mathrm{r}}$. □

Proof of Proposition 7. [§3.4.1 p.15] Recall from Section 2.3 that a cartless subtour starts and ends at the same position. Therefore, only circular subtours with a limited number of stops may arise. We differentiate between two cases: If $t^{\mathrm{c}}_{a,b} \leq t^{\mathrm{p}}_{a,b}$, partial tours without cart become superfluous. If $t^{\mathrm{c}}_{a,b} > t^{\mathrm{p}}_{a,b}$, when considering a single sub-aisle, transitions *III*–*VI* automatically result in slopes that denote the optimal path. The influence of cartless subtours only reduces the total travel time further in eligible slopes. For transition *I* the discussion is superfluous since it disconnects the path. Transition *II* remains as the only transition that does not yield a slope. However, for transition *II* traversing (a, b) by cart directly yields in lower overall tour duration: if (a, b) is traversed without traversing (b, a), the cart has to traverse (a, b) to complete the picker tour. Given $t^{\mathrm{c}}_{a,b} = t^{\mathrm{c}}_{b,a} > t^{\mathrm{p}}_{a,b} = t^{\mathrm{p}}_{b,a} \geq 0$, traversing (a, b) directly is always less costly than adding a circular subtour on foot of at least $2t^{\mathrm{p}}_{a,b}$ since $t^{\mathrm{c}}_{a,b} < t^{\mathrm{p}}_{a,b} + t^{\mathrm{c}}_{a,b} + t^{\mathrm{p}}_{b,a}$ always holds. Concluding, all slopes that are eligible for cartless subtours result automatically from the given transitions. Thus, further travel time reductions by cartless subtours can be identified during postprocessing. □

Proof of Proposition 8. [§3.4.1 p.16] Recall from Proposition 1 that the active interval of a pick list remains as short as possible in an optimal solution in order to not block the cart capacity longer than necessary. Traversing a sub-aisle without picking all items of at least one pick list results in an additional traversal, which translates into an extended active interval and hence, an unnecessarily blocked cart capacity. □

Proof of Proposition 9. [§3.4.1 p.16] We first prove Proposition 9 for two order lists, l_1 and l_2, and $\kappa = 2$. For each order list l_i, we can identify a subgraph H^{l_i} which allows to built a tour to collect all items in l_i. Focusing on the hulls ζ_{l_1}, ζ_{l_2} of H^{l_1}, H^{l_2} (see Figure 20), we consider three cases: *i*) ζ_{l_2} lies fully in ζ_{l_1} (Figure 20a): In this case, all mandatory sub-aisles in H^{l_2} can be visited through subtours or slopes while processing l_1. Hence, not processing l_1 and l_2 leads to additional traversals, which increase the total travel time in line with the rationale of Proposition 8. *ii*) ζ_{l_2} lies partially in ζ_{l_1} (Figure 20b): Since parts of ζ_{l_1} and ζ_{l_2} still overlap, the rationale of case *i* still holds and separate processing would yield additional traversals. *iii*) ζ_{l_2} and ζ_{l_1} (Figure 20c) overlap only in the depot vertex: In this case, a separate processing and a joint processing of l_1 and l_2 yield the same solution but a separate processing will never yield a better solution as all earlier or later order lists still connect in the depot vertex. For more than two order lists and $\kappa > 2$, the proof follows by induction, subsequently merging order lists. □

Proof of Proposition 10. [§3.4.1 p.16] With $d^+_i \neq d^-_i$, T^* remains either as a straight origin-destination path without cycles or contains one or more cycles. In both cases T^* must remain connected so that the only odd degrees relate to d^+_i and d^-_i. This holds since the traversal of each cycle must start and end in the same vertex since T^* remains an Eulerian cycle. In any infeasible case, additional odd degrees arise. □



(a) Fully overlaping hulls. **(b) Partly overlapping hulls.** **(c) Touching hulls.**

Figure 20: Three cases of overlapping hulls ζ_{l_1} (green) and ζ_{l_2} (brown).

D Merge procedure

The complexity of the merge procedure depends solely on the number of states, on the last layer of $\mathcal{L}^{\mathrm{b}}$ and $\mathcal{L}^{\mathrm{u}}$ as a pairwise comparison of states is necessary to merge partial path subgraphs from B_{ij} and U_{ij} in S. Even if the feasibility check can be conducted in constant time, the computational complexity of a straightforward merge is in $O(|\mathcal{L}^{\mathrm{b}}_l||\mathcal{L}^{\mathrm{u}}_l|)$ with l being the last layer in both graphs. To reduce the number of necessary comparisons, we exploit several characteristics that must hold in S to ensure feasibility:

i) Let $\mu = \sum_{i\in\{1,\dots,h\}} r_i$ be the total parity in S for a state $\alpha \in \mathcal{L}^{\mathrm{b}}_l$ and $\xi = \sum_{i\in\{1,\dots,h\}} r_i$ be the total parity in S for a state $\alpha \in \mathcal{L}^{\mathrm{u}}_l$. A merge of two partial path subgraphs can only be feasible if either μ and ξ are both even or are both odd.

ii) If μ and ξ are odd, an equal number of odd degrees must exist in $\mathbf{r}$. Moreover, $r_i \mod 2 = r'_i \mod 2$ must hold to ensure feasibility when merging $\alpha \in \mathcal{L}^{\mathrm{b}}_l$ and $\alpha' \in \mathcal{L}^{\mathrm{u}}_l$.

iii) If μ and ξ are even, $\exists i \in \{1,\dots,h\} | r_i > 0 \wedge r'_i > 0$ must hold to ensure feasibility when merging $\alpha \in \mathcal{L}^{\mathrm{b}}_l$ and $\alpha' \in \mathcal{L}^{\mathrm{u}}_l$.

We use these characteristics to divide $\mathcal{L}^{\mathrm{b}}_l$ and $\mathcal{L}^{\mathrm{u}}_l$ into subsets which reduce the number of necessary pairwise comparisons.

First, we use i) to divide $\mathcal{L}^{\mathrm{b}}_l = \overline{\mathcal{L}}^{\mathrm{b}}_l \cup \hat{\mathcal{L}}^{\mathrm{b}}_l$ into a set $\overline{\mathcal{L}}^{\mathrm{b}}_l$ with even μ and a set $\hat{\mathcal{L}}^{\mathrm{b}}_l$ with odd μ and analogously separate $\mathcal{L}^{\mathrm{u}}_l = \overline{\mathcal{L}}^{\mathrm{u}}_l \cup \hat{\mathcal{L}}^{\mathrm{u}}_l$. Limiting pairwise comparisons already reduces the computational complexity to $O(|\hat{\mathcal{L}}^{\mathrm{b}}_l||\hat{\mathcal{L}}^{\mathrm{u}}_l| + |\overline{\mathcal{L}}^{\mathrm{b}}_l||\overline{\mathcal{L}}^{\mathrm{u}}_l|)$.

Then, we use ii) to further divide $\hat{\mathcal{L}}^{\mathrm{b}}_l = \bigcup_{i\in\hat{\mathcal{I}}} \hat{\mathcal{L}}^{\mathrm{b}}_{l,i}$ and $\hat{\mathcal{L}}^{\mathrm{u}}_l = \bigcup_{i\in\hat{\mathcal{I}}} \hat{\mathcal{L}}^{\mathrm{u}}_{l,i}$ into subsets, each referring to states with a certain odd degree representation (cf. Example 1). Doing so, we derive $|\hat{\mathcal{I}}| = 2^h - 1$ subsets, which reduces the computational complexity to $O(\sum_{i\in\hat{\mathcal{I}}} |\hat{\mathcal{L}}^{\mathrm{b}}_{l,i}||\hat{\mathcal{L}}^{\mathrm{u}}_{l,i}| + |\overline{\mathcal{L}}^{\mathrm{b}}_l||\overline{\mathcal{L}}^{\mathrm{u}}_l|)$.

Finally, we use iii) to divide $\overline{\mathcal{L}}^{\mathrm{b}}_l = \bigcup_{i\in\overline{\mathcal{I}}} \overline{\mathcal{L}}^{\mathrm{b}}_{l,i}$ and $\overline{\mathcal{L}}^{\mathrm{u}}_l = \bigcup_{i\in\overline{\mathcal{I}}} \overline{\mathcal{L}}^{\mathrm{u}}_{l,i}$ into subsets, each referring to a representation in which a certain $r_i \in \mathbf{r}$ is greater than zero (cf. Example 3). Doing so, we derive $|\overline{\mathcal{I}}| = h$ subsets, which reduces the computational complexity to $O(\sum_{i\in\hat{\mathcal{I}}} |\hat{\mathcal{L}}^{\mathrm{b}}_{l,i}||\hat{\mathcal{L}}^{\mathrm{u}}_{l,i}| + \sum_{i\in\overline{\mathcal{I}}} |\overline{\mathcal{L}}^{\mathrm{b}}_{l,i}||\overline{\mathcal{L}}^{\mathrm{u}}_{l,i}|)$.

Utilizing these subsets to reduce the number of necessary pairwise comparisons, $O(\sum_{i\in\hat{\mathcal{I}}} |\hat{\mathcal{L}}^{\mathrm{b}}_{l,i}||\hat{\mathcal{L}}^{\mathrm{u}}_{l,i}| + \sum_{i\in\overline{\mathcal{I}}} ||\overline{\mathcal{L}}^{\mathrm{b}}_{l,i}||\overline{\mathcal{L}}^{\mathrm{u}}_{l,i}|)$ remains the worst case complexity in which all possible comparisons must be evaluated. To further speed up our merge procedure, we sort the states in each subset in increasing order with respect to their costs. This allows us to efficiently exploit potential merge moves and to exit the comparisons once a feasible solution is found and all succeeding pairs will show a higher potential objective value.

Example 3 (Odd degree representations in r) *We consider an arbitrary instance with $h = 3$ cross-aisles. Then, the population of different representations of odd degrees in $\mathbf{r}$ results to*

$$\begin{bmatrix} E \\ E \\ E \end{bmatrix}, \begin{bmatrix} O \\ E \\ E \end{bmatrix}, \begin{bmatrix} E \\ O \\ E \end{bmatrix}, \begin{bmatrix} E \\ E \\ O \end{bmatrix}, \begin{bmatrix} O \\ O \\ E \end{bmatrix}, \begin{bmatrix} E \\ O \\ O \end{bmatrix}, \begin{bmatrix} O \\ E \\ O \end{bmatrix}, \begin{bmatrix} O \\ O \\ O \end{bmatrix}$$

with E representing an even r_i and O representing an odd r_i. The first representation cannot arise if μ / ξ is odd, such that $2^h - 1$ representations remain.

References

Y. Asahiro, K. Kawahara, and E. Miyano. NP-hardness of the sorting buffer problem on the uniform metric. Discrete Applied Mathematics, 160:1453–1464, 2012.

K. Azadeh, R. M. B. de Koster, and D. Roy. Robotized and automated warehouse systems: Review and recent developments. Transportation Science, 53(4):917–940, 2018.

N. Boysen, S. Fedtke, and F. Weidinger. Optimizing automated sorting in warehouses: The minimum order spread sequencing problem. European Journal of Operational Research, 270(1):386–400, 2018.

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

N. Boysen, K. Stephan, and F. Weidinger. Manual order consolidation with put walls: The batched order bin sequencing problem. EURO Journal on Transportation and Logistics, 8(2):169–193, 2019b.

Y. A. Bozer and J. W. Kile. Order batching in walk-and-pick order picking systems. International Journal of Production Research, 46(7):1887–1909, 2008.

H. Cambazard and N. Catusse. Fixed-parameter algorithms for rectilinear Steiner tree and rectilinear traveling salesman problem in the plane. European Journal of Operational Research, 270(2):419–429, 2018.

F. Chen, H. Wang, C. Qi, and Y. Xie. An ant colony optimization routing algorithm for two order pickers with congestion consideration. Computers & Industrial Engineering, 66(1):77–85, 2013.

F. Y. L. Chin, Z. Guo, and H. Sun. Minimum manhattan network is NP-complete. Discrete & Computational Geometry, 45(4):701–722, 2011.

R. Cui, D. J. Zhang, and A. Bassamboo. Learning from inventory availability information: Evidence from field experiments on Amazon. Management Science., 65(3):1216–1235, 2019.

R. L. Daniels, J. L. Rummel, and R. Schantz. A model for warehouse order picking. European Journal of Operational Research, 105(1):1–17, 1998.

R. B. M. de Koster and E. S. van der Poort. Routing orderpickers in a warehouse: A comparison between optimal and heuristic solutions. IIE Transactions, 30(5):469–480, 1998.

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

J. Gallien and T. Weber. To wave or not to wave? Order release policies for warehouses with an automated sorter. Manufacturing & Service Operations Management, 12(4):642–662, 2010.

M. Goetschalckx and H. D. Ratliff. An efficient algorithm to cluster order picking items in a wide aisle. Engineering Costs and Production Economics, 13(4):263–271, 1988.

K. R. Gue, R. D. Meller, and J. D. Skufca. The effects of pick density on order picking areas with narrow aisles. IIE Transactions, 38(10):859–868, 2006.

R. W. Hall. Distance approximations for routing manual pickers in a warehouse. IIE Transactions, 25(4): 76–87, 1993.

M. Hanan. On Steiner's problem with rectilinear distance. SIAM Journal on Applied Mathematics, 14(2): 255–265, 1966.

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

S. Hong, A. L. Johnson, and B. A. Peters. Batch picking in narrow-aisle order picking systems with consideration for picker blocking. European Journal of Operational Research, 221(3):557–570, 2012a.

S. Hong, A. L. Johnson, and B. A. Peters. Large-scale order batching in parallel-aisle picking systems. IIE Transactions, 44(2):88–106, 2012b.

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

C. G. Petersen. An evaluation of order picking routeing policies. International Journal of Operations & Production Management, 17(11):1098–1111, 1997.

H. D. Ratliff and A. S. Rosenthal. Order-picking in a rectangular warehouse: A solvable case of the traveling salesman problem. Operations Research, 31(3):507–521, 1983.

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

K. J. Roodbergen, G. P. Sharp, and I. F. A. Vis. Designing the layout structure of manual order picking areas in warehouses. IIE Transactions, 40(11):1032–1045, 2008.

Statista. Annual retail e-commerce sales growth worldwide from 2014 to 2021. https://www.statista.com/statistics/288487/forecast-of-global-b2c-e-commerce-growt/, 2017.

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

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

F. Weidinger and N. Boysen. Scattered storage: How to distribute stock keeping units all around a mixed-shelves warehouse. Transportation Science., 52(6):1412–1427, 2018.

F. Weidinger, N. Boysen, and M. Schneider. Picker routing in the mixed-shelves warehouses of e-commerce retailers. European Journal of Operational Research, 274(2):501–515, 2019.

F. Wilcoxon. Individual comparisons by ranking methods. Biometric Bulletin, 1(6):80–83, 1945.

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