LUND UNIVERSITY

Order Picking Optimization as a Service

Oxenstierna, Johan

2024

Link to publication

Citation for published version (APA):
Oxenstierna, J. (2024). *Order Picking Optimization as a Service*. Computer Science, Lund University.

Total number of authors:
1

Creative Commons License:
Unspecified

General rights
Unless other specific re-use rights are stated the following general rights apply:
Copyright and moral rights for the publications made accessible in the public portal are retained by the authors and/or other copyright owners and it is a condition of accessing publications that users recognise and abide by the legal requirements associated with these rights.
• Users may download and print one copy of any publication from the public portal for the purpose of private study or research.
• You may not further distribute the material or use it for any profit-making activity or commercial gain
• You may freely distribute the URL identifying the publication in the public portal

Read more about Creative commons licenses: https://creativecommons.org/licenses/

Take down policy
If you believe that this document breaches copyright please contact us providing details, and we will remove access to the work immediately and investigate your claim.

LUND UNIVERSITY

PO Box 117
221 00 Lund
+46 46-222 00 00

Order-Picking Optimization as a Service

Johan Oxenstierna

LUND
UNIVERSITY

DOCTORAL DISSERTATION

Doctoral dissertation for the degree of Doctor of Philosophy (PhD) at the Faculty of Engineering at Lund University (LTH) to be publicly defended on 7 November 2024 at 1 pm, Department of Computer Science.

Advisors: Prof. Dr. Volker Krueger, Prof. Dr. Jacek Malec.

Abstract: Order-picking is one of the costliest processes in warehouses and we investigate how it can be optimized using Software as a Service (SaaS). First, we describe three specific order-picking optimization problems: The Picker Routing Problem (PRP), the Order Batching Problem (OBP) and the Storage Location Assignment Problem (SLAP). The PRP is a type of Traveling Salesman Problem (TSP) where we find a minimal-cost path for a vehicle assigned to pick a given set of products in the warehouse. The OBP is a type of Vehicle Routing Problem (VRP) where products are partitioned among a fleet of vehicles. We compute cost in the OBP by optimizing the PRP for each vehicle. In the SLAP, we assign or reassign storage locations of products such that costs in PRPs and/or OBPs are minimized. There are several choices regarding features and constraints for these problems, including digitization of warehouse rack layouts, zones, depot locations, dynamicity, product and vehicle characteristics, traffic rules and cost functions. In related work, there is little consensus on how to choose, classify and judge the importance of such features, leading to a lack of standards on data-driven benchmarking and experiment reproducibility. Before we propose optimization methods, we therefore examine choices and preprocessing of features to promote standardization. For our optimizers, we use heuristics and meta-heuristics. There exist publicly available heuristic solvers for the PRP capable of obtaining optimal solutions in a short CPU-time, but for the OBP and SLAP, optimal solutions often require an excessive amount of CPU-time. Consequently, we propose SaaS-suitable optimization techniques that balance between CPU-time, memory usage and cost minimization. We mainly rely on Monte Carlo methods, including Metropolis sampling and Nested Annealing, Sequential Minimal Distance (SMD), restart heuristics, cost approximation using the Quadratic Assignment Problem (QAP) and sub-optimal PRP optimization. Results show that costs found at early stages in optimization are often difficult to improve on, and that performance is sensitive to small changes in parameters and implementation in ways that are often difficult to foresee. For a SaaS which aims to provide optimization for multiple order-picking usecases, we therefore suggest a flexible workflow where various optimization methods are trialled and compared in sandbox environments. Data and results are shared in public repositories.

ISSN: 1404-1219

ISBN: 978-91-8104-215-3 (printed), 978-91-8104-216-0 (electronic)

Dissertation 75, 2024

LU-CS-DISS: 2024-02

Funding information: This work was partially supported by the Wallenberg AI, Autonomous Systems and Software Program (WASP) funded by the Knut and Alice Wallenberg Foundation.

Printed in Sweden by Media-Tryck, Lund University

Table of contents

List of Figures

Acknowledgements

First and foremost, I would like to express my deep gratitude to my academic supervisor, Volker Krueger, and my co-supervisor, Jacek Malec, for their unwavering support throughout my PhD journey. Volker, thank you for the countless insightful discussions, for guiding me in structuring my research and formulating research questions. Jacek, I am grateful for your help getting the work on the right track, especially at the early stages of the project.

I would also like to extend my thanks to my industrial supervisor, Lijo George, for helping me see the broader impact of my research and providing ongoing support. My appreciation goes to co-authors Louis Janse van Rensburg and Peter Stuckey for their valuable contributions on digitization and optimization. Thanks also to Björn Ekberg, Enys Mones, Michael Ashcroft and Nina Källgren for valuable support with software and project management.

I am incredibly thankful to Haorui Peng, Matthias Mayr, Alexander Dürr, Faseeh Ahmad, Momina Rizwan, Idriss Riouak, Gareth Callanan, Anton Risberg Alaküla, Rikard Olajos, Noric Couderc, Michail Boulasikis, Simon Kristoffersson Lind, Ayesha Jena, Hampus Åström, Esranur Ertürk, Gustaf Waldemarson, Peng Kuang, Leonard Papenmeier, as well as many others at the Computer Science Dept. whose camaraderie enriched this experience. Your friendship and the countless discussions we shared have been invaluable. Special thanks to Anders Bruce, Birger Swahn, Ulrika Templing, Heidi Adolfsson and Peter Möller for helping out with office equipment and administration.

Finally, to my family and friends, especially during the challenging times of the COVID-19 pandemic, thank you for your patience, support, and encouragement.

This work was partially supported by the Wallenberg AI, Autonomous Systems, and Software Program (WASP), funded by the Knut and Alice Wallenberg Foundation, whose generous funding is gratefully acknowledged.

Popular summary

"Too much of current warehousing practice is based on rules-of-thumb and simplistic ratios" [1].

Warehouse optimization can be undeniably complex, but also relatable. Everyday routines, such as the order with which one carries out shopping errands, the distribution of items between two shopping bags, or where to store the keys and wallet in the house, have formal analogues in warehouse research (Section 1.3, 1.4, 1.5, respectively). For humans, optimization of such routines is often subconscious. A supermarket shopper, as well as a warehouse picker, have a basic notion of the difference between an "efficient" and "inefficient" routine. But this does not mean that they always follow the efficient routine. Instead, they tend to follow a routine lying somewhere between the efficient and inefficient. It has been developed over years and carries a substantial amount of inertia. Why change that which already works?

The sub-optimality of existing routines is apparent in the domain of complex systems, such as Warehouse Management Systems (WMS). It is often easy to spot weaknesses in a WMS. It may have taken years of development and difficult decisions to get it working, followed by specific patches for multiple other systems that it connects with. Later, when core developers have left, a challenge may be to keep it updated and operational. When new features are needed or something in it fails, subsequent updates include brittle dependencies, erratically drawn connections between modules and spaghetti-code. This is what reality often looks like when consultants are brought in to work on a WMS.

The situation is not much better in the domain of warehouse research. While researchers can publish on topics that sidestep some of the intricacies of real WMSs, contributions cannot answer all questions that lay-persons may be pondering. Questions on warehouse operations, including order-picking optimization, often lack simple answers. Traditional conceptions are challenged by lean manufacturing, cloud-services and automation. Do we even

need warehouses or research on them? Could retailers not ship directly to customers? Or could we reduce warehousing to *cross-docking*, a method aimed at eliminating the need for storage, by instead timing the arrival of shipment trucks and moving products directly between them at docking stations?

We may never have good answers to such questions, but that does not mean that warehouse research is meaningless. In this dissertation, we begin by discussing the problem of what a basic warehouse is and the types of activities that commonly occurs within it. This type of standardization is an important driver not only for research, but for industry as well, as we can only compare operational quality if we have a stable fundament on which to base the comparisons on. A significant portion of the work in this dissertation concerns standardization of common features that are used to represent order-picking problems. Order-picking is widely considered one of the costliest activities in warehouses, and its optimization is both deserving and receiving an increasing amount of public attention.

For our quantitative work, we propose optimization methods for three optimization problems related to order-picking: The Picker Routing Problem (PRP), the Order Batching Problem (OBP) and the Storage Location Assignment Problem (SLAP). Optimization methods are proposed within the context of Software-as-a-Service (SaaS), where they are made accessible to warehouses over the cloud.

Terminology

AGV: Autonomous Guided Vehicle.

Batch: A set of orders. Usually, a batch is assumed to fit on a single vehicle.

Cross-docking: Transfer of products from a set of shipment vehicles to another set of shipment vehicles without moving products into long term storage.

Customer: There are three possible meanings relevant for the dissertation: 1. A client of a vendor. 2. A client of an order-picking optimization SaaS. 3. A location that needs to be visited (sometimes used in literature on the TSP and VRP).

Depot: The origin/destination location of a path (inside a warehouse in our case). This location is usually modeled as a vertex in a graph.

Deployment (cloud): Transferral of software from a local machine to a data centre. The purpose is often to make the software more accessible.

Distance matrix: A file containing the distances between locations.

Distribution centre: A type of warehouse focused on avoiding long storage for products.

FaaS: Function as a Service (Section 3.4.1).

IaaS: Infrastructure as a Service (Section 3.4.1).

Instance (benchmarking): Data which describes an example problem. For example, a basic TSP instance includes coordinates of a set of locations.

KPI: Key Performance Indicator, e.g., distance, time or operational cost.

Line-item: See product.

Makespan: time to finish a process or a set of processes.

OBP: Order Batching Problem. Generation of batches from orders such that some cost is minimized.

Operational horizon: see Makespan.

Optimization: The process of making something as good or effective as possible. An optimization problem is the problem of finding the best solution from all feasible solutions [2].

Order: A set of products.

Order-batching: A method of combining orders into batches.

Order-integrity: A constraint which requires that an order cannot be split into smaller components, i.e., that it must be picked by a single vehicle.

Order-line: Details on an order. Usually this includes product meta-data such as identifiers (SKUs) and corresponding quantities requested by a customer.

Order-picking: Retrieval of products from storage locations.

PAS: Pick-And-Sort batching. Items are first picked into a bin and then sorted into (shipping) boxes.

PaaS: Platform as a Service (Section 3.4.1).

Path: A sequence of vertices and edges where there are no repetitions of the same vertex. Synonymous to Hamiltonian path.

Pick-error: When a picker picks the wrong product in a pick-round.

Picker: See vehicle.

Pick-location: A location in the warehouse where a product can be picked.

Pick-round: Synonymous to pick-route/run, i.e., the sequence of pick locations visited by a single vehicle to pick a set of products.

Planning: the process of planning activities or events in an organized way so that they are successful or happen on time.

PRP: Picker Routing Problem. A Traveling Salesman Problem (TSP) set in a warehouse environment.

Problem instance: A digital description of a problem such that it can be optimized. A PRP instance, for example, can include information such as coordinates of locations and obstacles.

Product: A type of pickable item. The product is associated with meta-data, including a unique identifier (SKU) and information regarding location(s), dimensions, weight and quantity. A product with a quantity set to 10 means that there are 10 copies of it.

Put-away: The movement of products to storage locations.

Rack: A polygonal structure on which products are stored.

Replenishment: Increasing the quantities of products at their storage locations.

SaaS: Software-as-a-Service (Section 1.7.2).

Service time (for pick location): Time that a vehicle has to spend at a location to pick or deliver a product.

SKU: Stock Keeping Unit. The unique identifier/key of a product.

SLAP: Storage Location Assignment Problem (Section 1.5).

SMD: Sequential Minimal Distance. A way to measure the distance between two sets of coordinates. Can be used in clustering or the batching of orders.

Standardization: 1. Streamlining of process flows to achieve better cost efficiencies. 2. Forming agreement on terminology and features (including features used in data formats). The dissertation mainly focuses on the second definition.

SWP: Sort-While-Pick batching. Products are sorted into correct order containers during picking.

Tour: A sequence of vertices and edges where there are no repetitions of the same edge. Synonymous to trail.

TSP: Traveling Salesman Problem (Section 1.3).

Vehicle: Generic term for mobile units, trucks, forklifts, trolleys etc. that are loaded with products or orders during pick-rounds.

VRP: Vehicle Routing Problem.

Walk (in a graph): A sequence of edge and vertex visits.

Warehouse: A building primarily used for storage of products before distribution.

Warehouse Management System (WMS): "a complex software package that helps manage inventory, storage locations, and the workforce, to ensure that customer orders are picked quickly, packed, and shipped" [Bartholdi]. Basic features that WMS's support: appointment scheduling, receiving, quality assurance, put-away, location tracking, work-order management, order-picking, packing, consolidation, shipping, replenishment, wave management, yard management, labor management.

Wave (of orders): The available orders at a given time from which batches can be assembled.

Wave-picking: The term wave picking is used if orders for a common destination (for example, departure at a fixed time with a certain carrier) are released simultaneously for picking in multiple warehouse areas. Usually (but not necessarily) it is combined with batch picking [3].

1. Introduction

1.1 Order-Picking: A Warehouse Process

Figure 1: Typical warehouses as we see them from the outside [4]. On the fronts we can see docking stations for delivery vehicles.

A warehouse is a building primarily used for storage of products before distribution. Warehouses are important for two fundamental reasons [1]:

- *Supply and demand buffering*: Warehouses provide a buffer to meet quick surges in demand. Similarly, they provide a buffer for quick surges in supply. A bulk of products can be purchased for storage when vendors give price breaks, and then distributed to downstream customers at a later stage.
- *Transportation efficiency*: Warehouses are hubs for products usually located at transportation hubs. A warehouse normally receives bulk supply of certain kinds of products from a vendor. These products are redistributed with other kinds of products and delivered to downstream

customers. Without warehouses, many vendors need to send products to many customers, leading to transportation inefficiencies.

There are many types of warehouses. One way to categorize them is according to their types of stored products and served customers. Examples include retail, ecommerce and perishable goods warehouses. Vendors can also rent and carry out warehouse operations themselves in third-party logistics (3PL) warehouses. Warehouses can also be categorized according to how products are stored and picked. The most typical warehouse is a so-called *picker-to-parts* warehouse [3], which means that *pickers* move to pick *parts*, normally assumed to be stored on *racks* in some form of rack-*layout*. Typically, there is a single *depot*-location where pickers commence and terminate *pick-rounds*, i.e., movements to pick sets of products. After a set of products has been picked, it is checked and packed for delivery. In the alternative *parts-to-picker* warehouse, (mostly) stationary pickers receive products from autonomous vehicles, and mainly work with sorting, checking and packing efforts [5]. Examples of parts-to-picker warehouses include Robot-based Compact Storage and Retrieval Systems (RCSRs) and Robotic Mobile Fulfilment Systems (RMFSs) [6] (Figure 2).

Figure 2: The insides of warehouses can look very different. On the left is an example of a picker-to-parts warehouse with a conventional layout [7]. In the middle is an RCSR system and on the right an RMFS [6]. The pickers in the latter two cases are only involved with sorting, packing and checking products.

Some authors separate between *traditional*/*manual* warehouses and *smart* warehouses [8], [9], [10]. Smart warehouses fit within the frameworks of Industry 4.0 and Logistics 4.0 [9] and are deemed more up to date with the latest process designs, hardware and/or software to achieve better cost efficiencies.

For research purposes, common denominators between all warehouses are sought. Bartholdi & Hackman [1], for example, propose a chronology of five typical warehouse processes:

- *Receiving*: Products arrive at the warehouse in delivery vehicles (usually on pallets). The products are checked and staged for put-away.
- *Put-away*: The movement of products to storage locations (usually on pallets).
- *Order-Picking*: Retrieval of products from storage locations (Section 1.2).
- *Checking/Packing*: Checking and packing the products in one or multiple shipment containers (e.g., cartons).
- *Shipping*: Loading of the packed products in outbound delivery vehicles, followed by dispatch.

Some authors propose extensions or adjustments to the above process chronology. Kamali [8], for example, includes *replenishment* and *cross-docking* and Kembro & Norrman [9], regard the *packing* and *sorting* of products as separate processes. Geest et al. [10], also highlight the importance of *tracking/tracing* and *planning* in smart warehouses. Since there are several types of warehouses, there are also several ways to judge the importance of each process [11] (for a discussion on how to choose Key Performance Indicators (KPIs), see Section 3.1.8). Bartholdi & Hackman propose that a typical distribution of *operational expense* for *receiving*, *put-away* and *order-picking* is 10%, 15% and 55%, respectively. Worth noting is that the low expense of put-away versus order-picking assumes that products are moved to storage locations in larger quantities, e.g., fully loaded pallets, and then picked in smaller quantities. In many warehouses, however, vehicles carry the same quantity of products for both put-away and order-picking, leading to a more even distribution in expenses. Theoretically, put-away and order-picking can be considered the same but mirrored process, at least if we state generally that both involve the movement of products from one set of locations to another set of locations (origins to destinations). Order-picking is studied more extensively in the literature compared to put-away [12]. This can be attributable to the latter being considered equally or less complex, expensive and/or impactful for customer satisfaction (as it has a less direct impact on the time between placing an order and receiving it).

1.2 Order-Picking Optimization

"Order-picking is the most important process in most warehouses because it consumes the most labor" [1].

An order consists of one or several products and it can be viewed from two perspectives: 1. The customer, who selects a set of products from a vendor. 2. The warehouse provider, who picks and ships the order to the customer. There are cases where an order is picked and shipped from separate warehouses (including vendor's own warehouses), but research on order-picking generally works within the context of a single warehouse [1].

Due to its significant contribution to operational expense, warehouse managers are interested in *order-picking optimization* [1], [3], [11]. If we begin by approaching this topic from the perspective of a single warehouse-picking-vehicle, we can ask what the *shortest path* is for it to pick a set of products. We do not wish to have vehicles crisscross around the warehouse if this can be avoided, as this leads to more travel time and operational expense. If we have a set of products that is larger than what can fit on a single vehicle, we can ask how picking can be distributed among a fleet of vehicles, again with the basic idea of avoiding unnecessary travel. We may also ask whether it is possible to distribute products to vehicles such that the risk of customers getting orders with the wrong or missing products is reduced. Should orders be split between vehicles, or should we require that an order is always picked by a single vehicle? Another approach is to investigate the storage locations of products. Can we optimize storage locations to reduce order-picking travel costs? There are several other questions with regard to order-picking optimization, such as warehouse layouts, worker welfare and demand forecasting. But for meaningful research contributions, we need a narrow focus. Apart from delimiting our work to order-picking, we assume that a *picker-to-parts* system is used. Further, we optimize order-picking by working with three specific optimization problems: The Picker Routing Problem (PRP), the Order Batching Problem (OBP) and the Storage Location Assignment Problem (SLAP) (Sections 1.3, 1.4 and 1.5, respectively).

1.3 Picker Routing Problem (PRP)

The PRP is a Traveling Salesman Problem (TSP) set within a warehouse environment [13], [14] (Figure 3). As an example, we may have assigned a vehicle to pick 10 products (in one or several orders) distributed at various warehouse locations. The task of finding the shortest pick-round for these products can be formulated as a TSP.

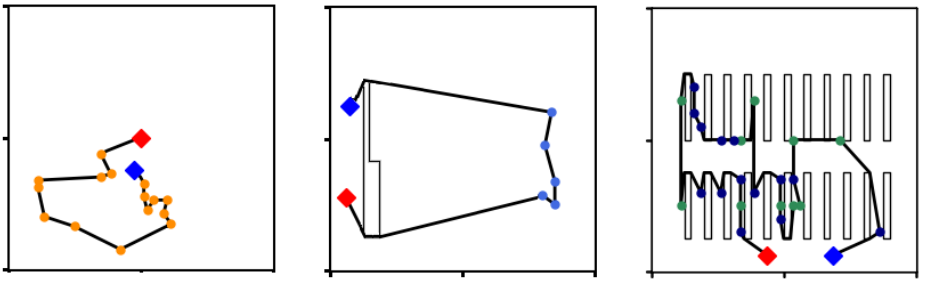

Figure 3: Typical (top-down) visualization of a solution to the TSP (left) and two PRPs, with one obstacle (middle) and several obstacles (right). Note that all three examples have different origin and destination locations (blue and red). In the rightmost example, we can also note that products may belong to separate orders (green or blue color codes), but that this does not affect the PRP solution. In the rightmost example, we can also see that we may have situations where an aisle must be entered and exited using a single obstacle corner location, showcasing that the PRP (in that case) is a form of *Steiner*-TSP.

In a typical warehouse, a set of racks or other obstacles obstruct straight paths between locations [15]. The obstacles may necessitate multiple visits to the same location, which results in a TSP version known as the *Steiner*–TSP [16]. For example, if the warehouse includes an aisle (between racks) that can only be entered and exited through a single location, we must use the location more than once to enter/exit the aisle. Similarly, multiple trips along the same path between locations may be needed.

In terms of graph theory [17], the most general form of a PRP solution is a *walk*, i.e., a sequence of vertices (representing locations) connected by a sequence of edges. It can then be classified as an *open walk*, where the origin (first) vertex is different from the destination (last) vertex, or a *closed walk*, where the origin and destination are the same (*single depot*).

For PRP optimization we need all possible shortest paths and costs between locations, and in Section 3.2.1 we describe how we produce these using

warehouse digitization. The way that this data is built and stored is important because it affects optimization CPU-time and memory requirements.

1.4 Order Batching Problem (OBP)

Order-picking is normally conducted by a fleet of vehicles with known carrying capacities, such as number of orders, products, weights, or dimension constraints. A batch denotes a set of orders, and in the OBP, a single vehicle is assumed to have enough capacity to carry one batch in its entirety. The OBP asks how one or several batches can best be generated without breaking vehicle capacity constraints. If we want to minimize travel distance, we try to generate batches whose union of products are located close to one another (Figure 4).

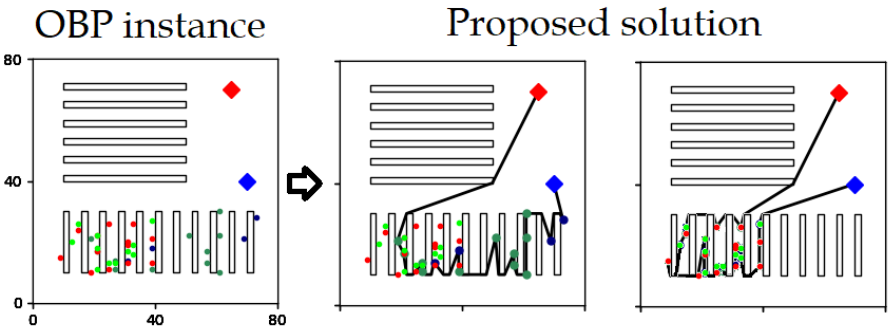

Figure 4: Example optimization instance (left) and proposed solution (right). We have four orders (represented in green, red, lime and blue products) and a vehicle capacity of two orders. The solution shows the selection of orders and pick-rounds for a first and a second vehicle.

Vehicle types and usage in the OBP can be separated into two categories [18]: 1. The *Sort-while-pick* OBP means that the products in an order are placed in an order-specific container on the vehicle (e.g., a bin or carton). 2. The *Pick-and-sort* OBP means that there is no order-specific container on the vehicle. In this case, the order is extracted from the vehicle after it has reached the destination.

Figure 5: Examples of vehicles used in warehouses. On the left is a trolley with 6 bins where each bin can be used to collect an order using sort-while-pick batching [19]. In the middle is a forklift carrying a pallet on which a set of products or orders is placed, e.g., using *pick-and-sort* batching [20]. On the right is a vehicle carrying a pod (a movable rack) [21], deployed in a Robotic Mobile Fulfillment Centre (a type of *parts-to-picker* warehouse) [6].

Some authors refer to a *Joint Order Batching and Picker Routing Problem* (JOBPRP) to underline that the cost of a batch is computed by optimizing its corresponding PRP [15], [22]. We use the term OBP instead of JOBPRP (see Section 1.7.1 for more on this).

1.5 Storage Location Assignment Problem (SLAP)

The SLAP asks where to store products in a warehouse, such that the amount of material handling (movement) costs is minimized [23]. The locations of products affect the distance that vehicles need to travel for order-picking. Therefore, the SLAP qualifies as a means to optimize order-picking. SLAP optimization is generally conducted periodically (e.g., once per month), and it requires some form of future-forecasting of order-picking to deduce whether changing locations of products can help reduce subsequent movement costs. We term this future-forecasted order-picking the *picking-log*, and we separate between two versions of the SLAP depending on how the picking-log is used during optimization: In the first version, future PRPs are optimized only in terms of their product locations, but not in terms of their order composition, i.e., batches are already pre-generated. In the second version, optimization includes both changing product locations as well as the order composition (i.e.,

batches are generated using OBP optimization). We term these versions the *TSP – based SLAP*[1] and the *OBP – based SLAP*, respectively.

While the OBP-based SLAP may seem more challenging than the TSP-based SLAP, there are several other features in the SLAP that can impact the level of challenge. One of these is *(re)assignment scenarios* [24]. In the most basic SLAP formulation, the task is to assign free locations to products that are newly arrived in the warehouse. Kübler et al. [24] call this the *empty storage location* scenario and it is equivalent to put-away. This can be compared to SLAP formulations that include *reassignments*, i.e., swapping locations between products that are already in the warehouse. Compared to the empty storage location/put-away scenario, reassignments are not mandatory to carry out, and therefore order-picking optimization savings, due to reassignments, must exceed the cost of carrying out the reassignments. In Section 3.2.4, we discuss how these reassignment scenarios can have a significant impact on optimization performance.

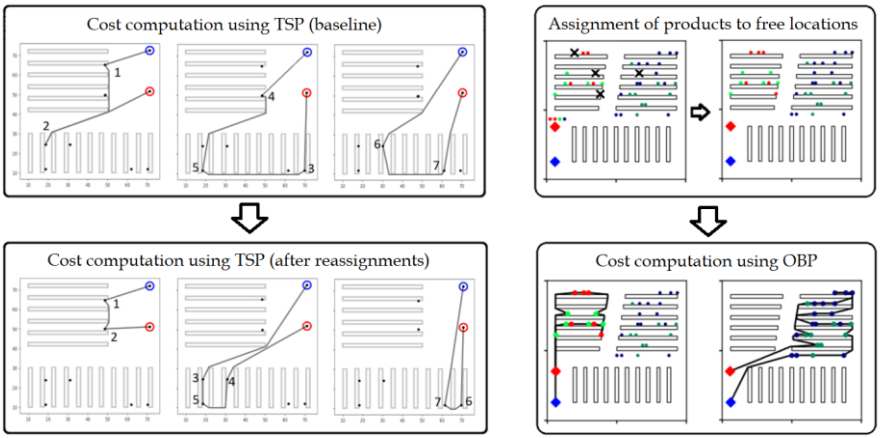

Figure 6: TSP-based SLAP in a reassignment scenario (left). The locations of seven enumerated products are changed to achieve lower TSP (PRP) distances. The incurred reassignment penalties are not visualized (See [25] for a full diagram). OBP-based SLAP in an empty storage location scenario (right). Here, four products are assigned free locations, followed by OBP optimization.

[1] While we could have used the term "PRP-based SLAP", we did not want to restrict our work on this version to a warehouse environment.

1.6 General Research Question

We have identified order-picking as a key process in warehouses and described the PRP, OBP and SLAP as problems with which order-picking can be optimized. We proceed to ask the following general research question:

How can an optimization software service be designed and provided for the PRP, OBP and SLAP?

Before we approach the general research question, we discuss challenges and context: Standardization (Section 1.7.1) and Software as a Service (SaaS) (Section 1.7.2). Standardization provides a framework within which we can place design choices for the optimization software. SaaS provides details on how the optimization software can be provided as a service. We then break down the general research question into five concrete research questions (Section 1.8).

1.7 Challenges and Context

1.7.1 Standardization

Standardization can have several meanings, and in operations research, it is commonly associated with how process flows can be streamlined to achieve better cost efficiencies [26], [27], [28]. According to Münsberg et al. [26], "standardization is essential to ensure a lean operating model", and Dotoli et al. [27], cover various standardization methodologies, including VSM, Genba, Jidoka, Kaizen, PDCA, Poka-Yoke, Muda-removal, KPI analysis and simulation. According to Shalley et al. [28], "standardization … [is] embodied in routines, repetition and variance reduction", and successful companies often achieve these by adhering to formal documents, including ISO 9000 and Six Sigma.

Standardization does not necessarily have to concern "lean operating modes" or "streamlined processes", however. In a more theoretical sense, it can be regarded as more aligned with the concept of *generalization* and the forming of agreement on the features used to formulate a problem (e.g., in order-picking optimization). We find significant gaps in research on PRP, OBP and SLAP optimization in this latter regard (Section 2.3). Lack of standards poses a

serious challenge to new research. Presentation of experimental results suffers without standards regarding the features used in experimentation.

Standardization of features builds on standardization of terminology. The warehouse terms used in this dissertation come from a large corpus of nomenclature that is continually revised in parallel with advances in technology. In related work on order-picking optimization, use of terminology is far from uniform, including extreme cases when authors refer to operational problems that are similar, but using different terms (Section 2.3). Without standards on terminology, difficult choices must be made when choosing terms. We have already presented examples of such choices, e.g., by using *OBP* instead of *JOBPRP* (Section 1.4). Contrary to the proponents of *JOBPRP*, we do not regard inclusion of "*PRP"* in the JOBPRP title as advantageous (specifically, because the title can be extended with various other acronyms as well). We also use *vehicle* instead of *picker* and *product* instead of *part*. This does not fit well with the *picker-to-parts* and *parts-to-picker* dichotomy (Section 1.1), since a separation between a picker and a vehicle is needed to explain *parts-to-picker*. The use of *picker* can be traced back to traditional warehousing, where the picker has been thought of as a human. But these days, *Autonomous Guided Vehicles* (AGVs), *mobile units*, *shuttles* and *pods* are deployed in ways that challenge this convention [6]. This is why we use *vehicle* in this dissertation, but it is not optimal either as it dehumanizes pickers if they are human. Selecting terms can be challenging both logically and ethically.

Standardization can be promoted through simulation and benchmarking [27], [29]. Publicly shared benchmark data is important for several reasons. Firstly, it makes it easier to independently reproduce published results. Secondly, it promotes competition to beat the state of the art on optimization problems, such as the PRP, OBP and SLAP. Benchmark data can also itself be regarded as an important descriptor of an optimization problem. Describing optimization problems in text, compact equations or pseudocode is not always easy, and benchmark data provide opportunities to approach PRPs, OBPs and SLAPs from a more data-centric perspective. Well-researched optimization problems, such as the Traveling Salesman Problem (TSP) and the Vehicle Routing Problem (VRP) (Section 2.3), are complemented by widely used benchmark data repositories [30], [31], [32], [33]. New researchers can therefore download datasets for these problems and start working with implementation at an early stage of a project. In the PRP, OBP and SLAP research communities, there are no well-established benchmark repositories, and new researchers need to decide whether the benchmark data that they find is adequate. There are many questions concerning how the PRP, OBP and

SLAP benchmark data should best be standardized. Should it be binary, text or JSON? Should it include distances between pairs of locations or information on obstacles? How many products, orders and pick-rounds should be used and how should they be distributed? How should we define vehicle capacity? Should we also work with a *dynamic* setting, where new information comes in through time, or a *static* setting, where all the information is assumed available? Answers to such questions are provided in terms of related work (Section 2) and proposed approaches (Section 3).

1.7.2 Software as a Service (SaaS) - Kairos Logic AB

This project is a collaboration between academia (LTH) and industry (Kairos Logic AB). Kairos Logic AB evolved from the Tenshi AI project at Sony Mobile Communications Inc. Apart from order-picking consultancy, Kairos Logic offers Software as a Service (SaaS) for PRP, OBP and SLAP optimization. The SaaS provides Application Programming Interfaces (APIs) and cloud service endpoints (URLs) where SaaS customers send HTTP PUT requests for optimization, followed by reception of optimized responses. The customer in this context is either a warehouse business or a Warehouse Management System (WMS) - provider, with examples such as Consafe Logistics, Landmark Group, Ongoing, Toyota Material Handling Group, Ahlsell, IKEA and Sony DADC (for convenience, we henceforth refer to the SaaS customer as a "WMS", since all warehouses we deal with operate some form of WMS). Before the endpoint is provided to the WMS customer, a Proof of Concept (POC) is carried out. Historic data, usually in the form of a spreadsheet of picking, is analyzed and processed using PRP, OBP or SLAP optimization. The results of the POC are presented in a consultancy meeting. After the meeting, the WMS customer can choose to proceed further with more case studies and/or access to live testing of the service.

This dissertation does not delve into the business potential of this type of optimization SaaS. Rather, it concerns order-picking optimization *within the SaaS context*. This is important to highlight, as it explains several choices with regard to how the general research question (Section 1.6) can be approached. In brevity, the proposed optimization methods need to be relatively fast, standardized and easily maintained so that integration is simple and works for several customers. In Section 3, we discuss how various industrial contexts affect architectural choices. In order to handle optimization requests efficiently, architectural choices are important not only for the optimization service in isolation, but also for warehouse digitization. We also explain why

the optimization service is deployed using Platform as a Service (PaaS) or Infrastructure as a Service (IaaS) instead of Function as a Service (FaaS).

We provide an example of how the SaaS architecture relates to the standardization topic discussed in Section 1.7.1. A challenge to this specific SaaS optimization business is whether the same service should be offered to multiple warehouses or if it should be tailored for specific warehouses. A uniform service is desirable from the standardization perspective, but it cannot be fully achieved if the optimization business is customer-driven: There are dozens of versions of PRPs, OBPs and SLAPs (Section 2.2 and 2.3), and these versions translate into a large variety of customer requirements. In practice, the SaaS-business is often unable to satisfy all requirements of a new customer. The unfulfilled requirements are usually brought to light during or after the POC, and the customer commonly demands them to be fulfilled to proceed further. If the SaaS provider agrees to fulfil the requirements, they usually do not cause issues if they are implemented as extensions to the existing software stack. Worse is if the requirements are of the modification type, where some previously standardized functionality must be changed. Such modifications can, in the end, lead to improvements to the existing software stack, but it can also worsen it. In some cases, requirements are not well-defined or unsuitable for optimization. This is unfortunately quite common because of complexities involved in PRP, OBP and SLAP optimization, misunderstandings and/or miscommunications. Examples of this are discussed in Section 4.

1.8 Concrete Research Questions

We now formulate five concrete research questions that we aim to answer in this thesis:

1. *How can SaaS-suitable PRPs and OBPs be formulated? This includes feature standardization and CPU-time and memory needs in warehouse digitization.*
2. *Building on the requirements outlined in 1, how can PRP and OBP optimization algorithms be designed and benchmarked?*
3. *How well do the proposed optimization algorithms in 2 perform against alternative algorithms and are there possible improvements?*
4. *How can OBP-optimization be utilized within an OBP-based SLAP optimization algorithm?*

5. *What are the challenges with the OBP-based SLAP and are there alternative ways to optimize the SLAP?*

In Section 1.9, we give a summary of how each question is approached in a corresponding paper (numbered 1-5). We include pointers to subsequent sections where the main topics in the papers are discussed.

1.9 Summary of Papers

Paper 1: Formulation of a Layout-Agnostic Order Batching Problem.

Oxenstierna, J., van Rensburg, L. J., Malec, J., & Krueger, V. (2021, June). In International Conference on Optimization and Learning (pp. 216-226). Springer International Publishing.

This paper addresses limitations in current research on PRP and OBP optimization and feature standardization, particularly with regard to warehouse layouts. Building on existing work, we formulate an Order Batching Problem (OBP) where optimal PRP optimization is internalized. In the *layout-agnostic OBP*, we do not make assumptions regarding how racks or other obstacles are laid out on a two-dimensional plane. For optimization to work in this scenario, certain datastructures, such as the distance matrix, must be prepared beforehand in a warehouse digitization process. In our experiments, we digitize nine warehouses with various arrangements of polygonal obstacles and investigate CPU-time and memory requirements. CPU-times and memory requirements are relevant for a SaaS business because they incur costs from a cloud service provider. Outline of key topics in the paper:

- *Identification of current gaps in related work regarding features used in PRP and OBP optimization and feature standardization (Section 2.2, 2.3)*
- *Advantages and disadvantages of layout-agnostic order-picking optimization (Section 3.1.1).*
- *Efficiency and scalability with regard to warehouse digitization and SaaS deployment (Section 3.2.1, 3.4).*

Paper 2: Layout-Agnostic Order Batching Optimization.

Oxenstierna, J., Malec, J., & Krueger, V. (2021, September). International Conference on Computational Logistics (pp. 115-129). Springer International Publishing.

Building on the previous paper, we introduce a heuristic OBP optimization algorithm, Single Batch Iterated (SBI). It uses the Concorde TSP solver for optimal PRP cost evaluation and the Sequential Minimal Distance (SMD) heuristic for batch construction. Experiments show that the algorithm is competitive against the state-of-the-art on an existing benchmark dataset. A new benchmark dataset for various layouts and constraints is publicly shared together with proposed solutions. Outline of key topics in the paper:

- *Advantages and disadvantages of including optimal PRP optimization in an OBP optimization algorithm (Section 2.4.2, 3.2.2).*
- *The performance of the proposed OBP optimization algorithm against the state-of-the-art (Section 3.2.3).*
- *Questions regarding how new benchmark datasets be constructed to promote standardization (Section 3.3).*

Paper 3: Efficient Order Batching Optimization using Seed Heuristics and the Metropolis Algorithm.

Oxenstierna, J., Malec, J., & Krueger, V. (2022). Springer Nature Computer Science, 4(2), 107.

In this paper, the previously developed SBI algorithm is improved and tested one-on-one against Metropolis Batch Sampling (MBS), a type of Markov Chain Monte Carlo (MCMC) algorithm. On existing benchmark data, SBI is found to be superior, especially for larger problem instances. This result is attributable to its effective use of heuristics that help it navigate large search spaces. Outline of key topics in the paper:

- *The main heuristics used in SBI and their effect on optimization performance (Section 3.2.3).*
- *Reasons for SBI's strong performance against MCMC (the MBS algorithm) (Section 3.2.3).*

Paper 4: Storage Assignment using Nested Metropolis Sampling and Approximations of Order Batching Travel Costs.

Oxenstierna, J., Malec, J., & Krueger, V. (2024). Springer Nature Computer Science, 5(5), 477.

An OBP-based Storage Location Assignment Problem (SLAP) is first formulated, with the core idea that proposed changes in location assignments are evaluated using OBP optimization. For optimization of the OBP-based SLAP, a nested Metropolis algorithm is used. It incorporates the SBI optimizer, as well as a Quadratic Assignment Problem (QAP)-based cost approximator. Experiments include tests of the QAP model in isolation, as well as tests of the overall SLAP optimizer on a new and publicly shared benchmark dataset. Outline of key topics in the paper:

- *Formulation of the OBP-based SLAP (Section 3.2.4).*
- *Investigations of a QAP model's ability to predict OBP optimization costs (Section 3.2.4).*
- *Performance of the nested Metropolis algorithm when it includes or excludes the QAP model (Section 3.2.4).*

Paper 5: Optimization of the Storage Location Assignment Problem using Nested Annealing.

Oxenstierna, J., van Rensburg, L. J., Stuckey, P. J., & Krueger, V. (2022, February). International Conference on Operations Research and Enterprise Systems (pp. 220-244). Part of the Communications in Computer and Information Science book series (CCIS, volume 1985).

In this paper, an alternative SLAP model is first formulated: OBP-optimization is replaced with PRP-optimization, with the argument that this is more suitable for standardization. The SLAP is optimized using the Concorde TSP solver and Nested Annealing. In order to improve the computational efficiency of the algorithm, various heuristics are tested. Results show that restart heuristics and sub-optimal PRP optimization are especially useful. A new benchmark dataset is publicly shared. Outline of key topics in the paper:

- *Formulation of the TSP-based SLAP and its strengths and weaknesses compared to the OBP-based SLAP. This includes discussions on standardization and integration challenges (Section 3.2.4, 3.4.2).*

- *The impact on computational efficiency of restart heuristics and sub-optimal PRP optimization in SLAP optimization (Section 3.2.4).*

1.10 Disposition

In the section on related work (Section 2), we first provide a background on operations-research, computer-science and warehouse-science (Section 2.1), followed by descriptions of key features and terms used in order-picking optimization (Section 2.2). We continue with a discussion on the standardization challenge (Section 2.3), first with regard to features and terms (Section 2.3.1), followed by an industrial context (Section 2.3.2). In Section 2.4, we study related work on PRP, OBP and SLAP optimization methods. In the Approach section (Section 3), we begin by discussing and justifying our selection of features in PRPs, OBPs and SLAPs (Section 3.1), followed by optimization methods (Section 3.2). We then discuss benchmarking (Section 3.3) and industrial context (Section 3.4). This is followed by discussions of industrial projects outside the main scope of this dissertation (Section 4). Section 3.4 and 4 can be regarded as extensions of Section 1.7, as they strengthen the argument that standardization and SaaS–specific considerations provide important challenges and context for the project as a whole. We end with a conclusion (Section 5).

2. Related Work

2.1 Background

The related work originates in the domains of *operations-research*, *computer-science* and *warehouse-science*. In this section, we motivate why it is suitable to use these domains when approaching the main research question (Section 1.6).

According to Morse et al. [34], operations research "is an applied science utilizing all known scientific techniques as tools in solving a specific problem". We use scientific techniques from computer-science and warehouse-science to provide answers on how order-picking optimization can be designed and provided as a service. Operations-research is particularly suitable since it fits well with the project's combined academic – industrial context (Section 1.7.2). According to Morse et al., a core element of operations-research is the separation between a *research worker* and an *executive*. The role of the research worker is to "provide the executive with a quantitative basis for decision" [34]. As a collaboration between a university and a company, we have a concrete example of an executive, the CEO of the company, whom we need to convince that our quantitative research can be used as a basis for decision.

Operations-research has traditionally relied on mathematics for the "quantitative basis", but computer-science is similarly a valid option [35]. Apart from a stable grounding in mathematics [36], computer-science connects well with the *service* aspect of the main research question. Both the proposed optimization service and the warehouse systems it connects with operate on computers. Literature on computer-science sub-topics, such as memory requirements, computational times and efficiency in optimization, is therefore relevant. Since the optimization service is cloud-based, computer-science is used to provide answers on cloud infrastructure and networking.

Compared to operations-research and computer-science, *warehouse-science* [1] is a small domain. Authors on order-picking optimization mainly publish

in journals with names derived from “operations-research”, “industrial-engineering” or “production-research” rather than warehouse-science [12]. There are also journals on “inventory management”, “supply chain” and “logistics”, within which warehouse-science can be considered a subset [37]. We primarily use warehouse-science to provide context for the operational setting, looking particularly at terminology and descriptions of the order-picking process. This does not mean that warehouse-science itself should be categorized as “descriptive”. Bartholdi & Hackman [1] define warehouse-science in a fashion more akin to operations-research: “the emphasis … is on developing methodology to optimize warehouse operations”.

As with operations-research, we could regard computer-science as a scientific technique *within* warehouse-science. But in this dissertation, we treat computer-science more independently. For example, we include discussions on why certain methods are deemed unsuitable for standardized order-picking optimization. This particularly concerns Machine Learning (ML) and optimal solutions for the OBP and the SLAP (Section 2.4). We also look at how an order-picking optimization SaaS can be deployed on the cloud, a topic not particularly well-researched generally. The SaaS architecture can be used to investigate ideas on future warehousing. One such hypothesis is that warehouses are going to become smaller and more flexible in the future [1]. We can ask, for example, whether the SaaS architecture is well-suited for small and flexible warehouses.

2.2 PRP, OBP and SLAP Features

The PRP, OBP and SLAP are part of a large family of NP-hard combinatorial optimization problems. They have corollaries in the Traveling Salesman Problem (TSP) (Section 1.3), the Vehicle Routing Problem (VRP) [22] and the Location Routing Problem (LRP) [38]. There are many versions of these problems, most of whom have their primary usecases outside of warehouses [12], [23], [39], [40]. Since PRPs, OBPs and SLAPs occur inside warehouses, research on them tend to include warehouse-specific features. Below, we go through how features are commonly named, designed and used in PRP, OBP and SLAP optimization:

- *Layout*: In the majority of literature on order-picking, it is assumed that the warehouse uses a *conventional layout* [12], [22], [41], [42]: Racks are arranged in Manhattan style blocks separated by aisles and

blocks of racks are separated by cross-aisles. There are several other possible layouts [5], [43], [44], [45]. Masae et al. [12], distinguish between *conventional*, *non-conventional* and *general* layouts. It is usually assumed that order-picking takes place on a single floor, which means that the layout is defined on an xy Cartesian plane.

- *Locations and zones*: It is often assumed that a single product is stored at each defined location in the warehouse [24]. In some cases, a single product is stored at multiple locations (Section 4.1). In other cases, multiple products are stored at a single location. This latter case may be useful to reduce the size of a digital model of a warehouse, as well as order-picking optimization complexity [14], [46]. Defining a surjective relationship between many products to fewer locations (within a specified area) is similar to a *zone* in the warehouse [18]. Zones are usually implemented as a form or "load-balancing" of order-picking throughout the warehouse [47], or to distinguish frequently picked products from less frequent ones [48]. For example, an *ABC zoning policy* can be used, where the A zone includes the most frequently picked products, followed by B etc. [24], [49], [50]. Various other zone arrangements exist: Garfinkel [18], for example, studies scenarios with 10 – 40 zones.

- *Depots*: Usually, a warehouse is assumed to have a single shared origin and destination for vehicles [3]. Some authors refer to this as a *fixed* or *central depot* [51]. For cases where there are different origin and destination locations, the names *variable depots*, *open-trip*, *multi-depot TSP* and *Dial-a-ride problem* are used [38], [51], [52]. These scenarios can occur in warehouses where products are brought to multiple docking stations. There are also cases when different types of vehicles have different docking stations.

- *Dynamicity*: If time-based features are needed to describe and optimize a problem instance (e.g., a PRP, OBP or SLAP), it is dynamic [33], [53]. If there are no such features, it is static. The time-based features can take several forms, such as live information on vehicle locations (used to avoid traffic congestion, for example), or soft or hard time constraints on when products or orders need to be picked [52]. For the SLAP, dynamic seasonal popularities of products may be used.

- *Stochasticity*: We discuss stochasticity with regard to how PRPs, OBPs and SLAPs can be digitally represented in simulated problem instances. Problem instances can be separated into *pre-generated* and

randomly-generated [54]. A random process can be used to generate data, such as vehicle origin and destination locations, travel times and order sizes in both of these [33]. But the randomly-generated type does not provide all the static data needed to run experiments [33]. Instead, the randomly-generated instance includes information on how to use a random process to generate static data [52].

		Information quality	
		Deterministic input	Stochastic input
Information evolution	Input known beforehand	Static and deterministic	Static and stochastic
	Input changes over time	Dynamic and deterministic	Dynamic and stochastic

Figure 7: Dynamicity and stochasticity [32]. Benchmark instances can be constructed using either one of the four combinations.

- *Order-integrity constraint*: This constraint means that orders cannot be split between vehicles. It follows that the *capacity* of the vehicle must always be large enough to carry at least one full order in its entirety. In reality, however, an order may be too large to fit on a vehicle, necessitating it to be split (in Section 3.1 we describe how we handle this scenario). Order-integrity is usually motivated as a means to avoid additional sorting efforts after picking [52]. It is generally used in the OBP and less so in alternative picking methods, such as wave-picking [3].
- *Constraints concerning product characteristics*: Products may be hazardous, in need of cooling, specialized picking and/or storage [1]. Light products may need to be picked after heavy ones (so that they can be placed on top) using precedence constraints [55].
- *Traffic rules*: When vehicles intersect each other's paths, they may cause traffic congestion. Traffic congestion is often considered important in warehouses, as it serves as a motivation for imposing one-directional traffic rules and/or certain PRP optimization algorithms [3], [15]. In the SLAP, it also serves as a motivation to prevent too many products from being assigned locations close to a depot area (since this may lead to congestion) [56].

- *Capacity constraints*: Example capacities include number of orders, products, xyz dimensions and weight. For the OBP, these capacities can concern vehicles (e.g., weight that a vehicle can carry). For the SLAP, they can concern locations (e.g., product quantity that can be stored at a single location).
- *Cost function*: Cost in PRP, OBP and SLAP optimization is most often measured in distance [3], followed by travel times and times based on other features. The time-based costs are more prevalent in literature on the OBP and the SLAP, since both are more involved with stationary sorting efforts. For example, in the OBP, the time needed to sort and check orders after they have been picked can be included [57]. In the SLAP, administrative times needed to carry out optional product-location reassignments can be included [24]. Gibson & Sharp [58], suggest four alternatives to measure distance in a warehouse: *Rectilinear* (Manhattan), *Euclidean*, *Chebyshev* and *aisle*. The latter distance is computed by using known dimensions of aisles and cross-aisles.

2.3 PRP, OBP and SLAP Standardization

2.3.1 Feature Standardization

There is little consensus regarding the relative importance and relationships between the features listed in Section 2.2. Even when excluding *parts-to-picker* systems, there are many possible combinations of features when formulating order-picking problems [12], [23], [40], [59]. When defining a PRP, OBP or SLAP, we can assume that we must choose from some features listed in Section 2.2. If we need to make a binary decision for seven features, we get $2^7 = 128$ possible combinations. This number can be compared against number of papers referred to in review papers on order-picking problems by Masae et al. [12], Charris et al. [23], Li et al. [60] and Pardo et al. [61]. They include 149, 71, 172 and 125 papers, respectively. The number of papers on any specific version of the PRP, OBP or SLAP, based on a combination of features in Section 2.2, is low and this poses a standardization challenge. Pardo et al., for example, propose that the OBP can be separated into 36 specific versions, 18 of which have never been published on [61].

The standardization challenge is also visible through differences on optimization choices, taxonomy and naming of problems. Regarding optimization choices, Mantel et al. [62], claim that "extensive batching extinguishes the effect of a clever slotting [SLAP] strategy", whereas Kübler et al. [24], include batching in their SLAP strategy to reach "significant performance improvements". Regarding taxonomy, Charris et al. [23], Li et al. [60] and Masae et al. [12], propose different diagrams to describe relationships between features such as layouts, storage conditions, depots and picking methodologies (Figure 8, 9 and 10, respectively). For example, Charris et al. have an arrangement with "SKU-department assignment", "zoning" and "storage location assignment" under "storage", whereas Li et al. divide "storage assignment" into "random", "closest open location", "dedicated", "full turnover" and "class based". Under "batching", Li et al. put "proximity batching" and "time window batching", whereas Charris et al. divide it into "batch size" and "order batch assignment". Charris et al. include "sorting" in their diagram, but this is not included by Li et al. or Masae et al. Meanwhile, Li et al. include six "routing methods" and Masae et al. include a box on "depots", neither of which are present in any other diagrams. In summary, there are many possible ways to build taxonomies for combinations of features in order-picking problems.

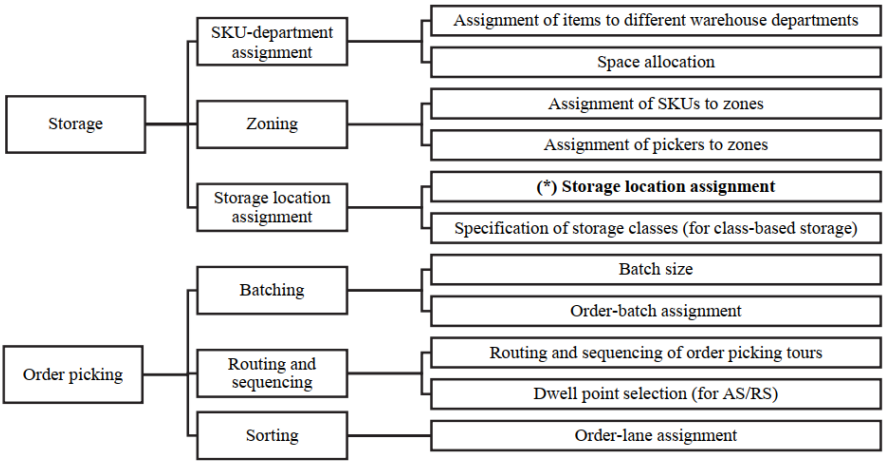

Figure 8: Taxonomy on storage and order-picking by Charris et al. [23].

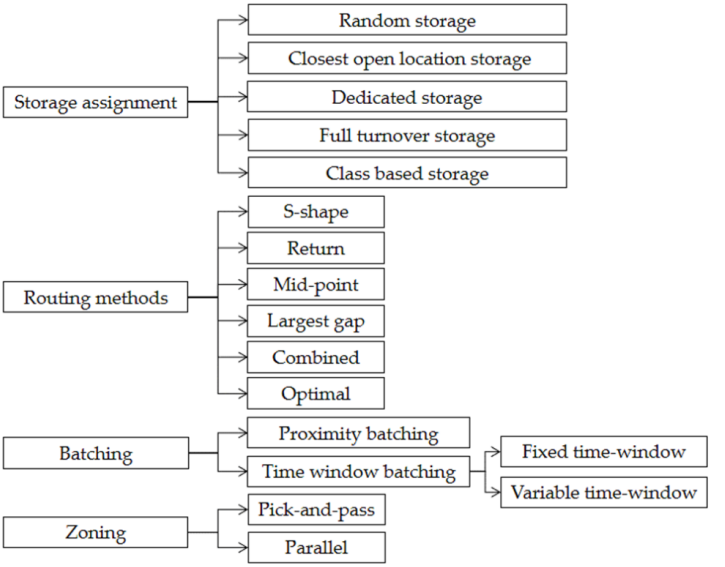

Figure 9: Taxonomy on order-picking by Li et al. [60].

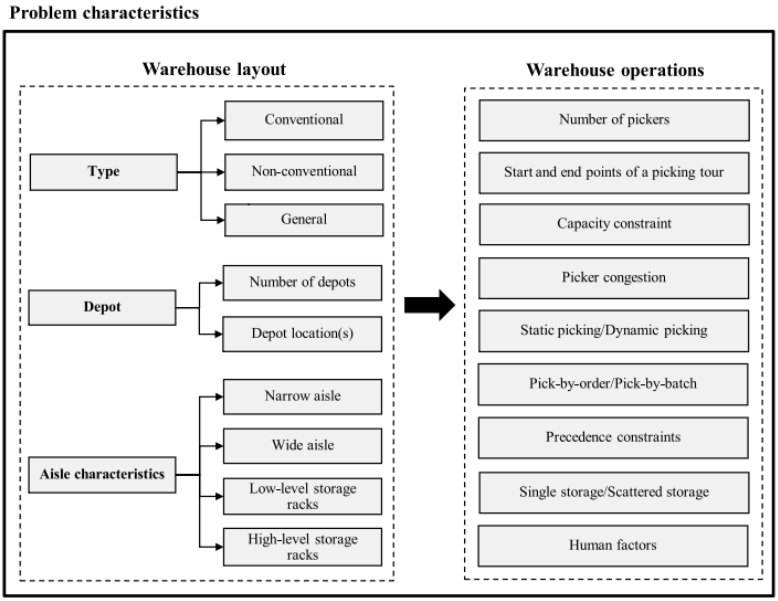

Figure 10: Taxonomy on order-picking routing by Masae et al. [12].

Regarding naming of problems, Mantel et al. [62], Kim & Smith [63] and Jahani [49] refer to a "slotting problem", Boysen & Stephan [64] to a "Product Location Problem" and Garfinkel [18] to a "correlated storage assignment problem", when they could have used the "SLAP" term [23], [24], [65], [66]. We can also observe a lack of references in literature on the OBP and SLAP to literature on their respective counterparts in the Vehicle Routing Problem (VRP) and the Location Routing Problem (LRP). We could, for example, claim that the OBP is a type of VRP and then use literature on the VRP when working on an OBP. An OBP is equivalent to a common VRP if all the orders contain one product each that never share the same location [23], [67]. A VRP, on the other hand, is equivalent to an OBP if it is of the following kind: A *Steiner Clustered VRP with Soft Cluster Constraints* (Steiner SoftCluVRP). The SoftCluVRP was introduced by Hintsch & Irnich [68], but it lacks the requirement that clusters can share locations ("customers" using their terminology), hence the addition of the *Steiner* prefix. Different terms are used to describe the same features in the respective research communities: An OBP "location" is (often) equivalent to a VRP "customer", an OBP "order" is equivalent to a CluVRP "cluster", the "soft cluster" constraint is equivalent to the "order-integrity" constraint.

In the case of the SLAP and LRP, there is also a limited exchange of knowledge. If we look at review papers on the SLAP [23] and the LRP [38], respectively, we note that neither refers to both problems, despite their similarities. The main difference between the two is that the LRP is concerned with finding the best locations for depots, whereas the SLAP is concerned with finding the best locations for products. But the impact of changing a depot location and a product location is very similar: Generally, we need to solve a new routing instance to evaluate the impact of conducting changes in locations, regardless of whether they are for depots or products.

The evolution of parallel terminologies can also be motivated. A warehouse has certain differences to the various outdoor usecases that the TSP, VRP and LRP are usually concerned with. An OBP formulation may be formally equivalent to a *SteinerSoftCluVRP*, but they are expected to be applied in different environments. Apart from considerations concerning warehouse rack-layouts, vehicles in warehouses tend to be smaller and operating on smaller timescales compared to outdoor counterparts. The domain of warehouse vehicles also changes quickly. A warehouse AGV taxonomy provided by Azadeh et al. [6] is not fixed and can be expected to change as technology evolves. Hence, even if the "inside warehouse" or "outside

warehouse" domains are similar currently, they may diverge in the future, and this motivates semantic separations.

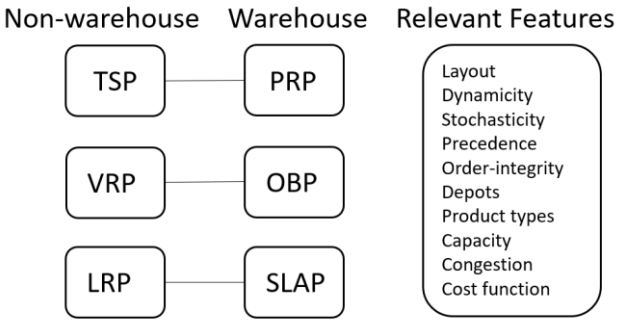

Figure 11: The PRP, OBP and SLAP have similar problems outside of warehouses. But this does not necessarily mean that semantic separations are unmotivated.

In terms of features used in benchmarking for order-picking problems, there exists no well-established standards. For the OBP, instances include Foodmart [22] and HappyChic [69], both of which describe niche usecases. In Foodmart, each vehicle carries bins and orders may be split between them. This adds a bin-packing problem on top of the OBP and renders their version a hybrid between pick-and-sort and sort-while-pick batching. HappyChic is tied to a certain layout where vehicles must move uni-directionally around a conveyer belt of a certain design. In the VRP research community, there are several well-known instance repositories, such as the Solomon, Christofides, Taillard, Augerat et al., Fisher and Kilby instances [30], [31], [32], [33]. For the PRP and SLAP, we are not aware of workable instances. For the TSP, instance repositories include TSPLIB [70] and Cook [71]. There are benchmark instances for the LRP, but comparisons between published results are insufficient. Nagy [40], for example, claims that only four published papers on LRPs include comparisons of results to those of other authors.

2.3.2 Industrial Context

There does not exist much prior research on optimizing PRPs, OBPs and SLAPs in the SaaS format described in Section 1.7.2. In this section, we briefly discuss this topic from the perspective of the WMS, as well as the SaaS provider.

Cloud-based WMSs are widely considered superior to more traditional alternatives [72], [73], [74], [75] , but they come with certain caveats, and it is

not self-evident that order-picking optimization is something that a WMS wants to outsource to a SaaS-provider. From the WMS's perspective, connection to a SaaS enforces a high level of modularity, leading to decentralization [1], [76]. This can be advantageous because the engineers of the WMS only have to write a module which connects to a SaaS. Any development, maintenance and scaling of the PRP, OBP and SLAP optimizers are handled by the SaaS providers. But since the WMS developers lose some degree of control over their system, it can be disadvantageous in terms of security and legal ramifications [77].

For the WMS team to be interested in using the SaaS, it also needs to be set up in a way such that it is easy to integrate. Apart from reliability and adhering to RESTful API principles [78], [79], computational efficiency is also relevant. The WMS team needs to have information about the amount of cost savings that can be achieved within a known amount of time. The allowed wait-time for a request to be answered can have effect on the choice of deployment type for the SaaS, e.g., Infrastructure as a Service (IaaS), Platform as a Service (PaaS) and Function as a Service (FaaS) [80], [81]. There are also trade-offs between holding warehouse-related files in memory in a PaaS or IaaS, making them instantly available for optimization, versus a FaaS with a cold-start, where some time is spent loading files into memory before optimization can begin [81]. In Section 3.4, we provide further details on deployment options.

2.4 Optimization

2.4.1 Problem Formulations

We now study how PRPs, OBPs and SLAPs have been formulated in the literature. We only look at formulations which use distance as a cost function. Although Masae et al. [12] and Charris et al. [23] point to several cost alternatives, we follow Koster et al.'s [3] argument that distance is of primary importance. This is mainly because alternatives *include* distance functions, as well as other functions (for a time-based cost prediction, for example, the expected distance to be travelled is one of the main inputs [41]).

We begin by formulating a distance-minimizing PRP as a general TSP. One possible way to formulate a TSP is through two-index Integer Programming (IP) [82], where there are n vertices (one of which is a single origin/destination, i.e., depot) and all pairwise distances between the vertices are known:

$$min \sum_{\substack{0 \le i \\ i \ne j}} \sum_{\substack{j \le n \\ j \ne i}} d_{ij} x_{ij}, d_{ij} \in \mathbb{R}^{+}, x_{ij} \in \{0, 1\}, \quad (1)$$

s.t.

$$\sum_{\substack{i=0 \\ i \ne j}}^{n} x_{ij} = 1,\ j = 1, \dots, n, \quad (2)$$

$$\sum_{\substack{j=0 \\ j \ne i}}^{n} x_{ij} = 1, i = 1, \dots, n, \quad (3)$$

$$u_i - u_j + p x_{ij} \le p - 1, 1 \le i \ne j \le n. \quad (4)$$

The binary x indicates whether a certain edge ij (between vertices i and j) is traversed. The distance between vertices i and j is d_{ij}. Constraints (2) and (3) ensure that each vertex is connected with exactly two edges (one incoming and one outgoing). The inequality in Equation 4 is needed to ensure that all of the vertices are connected in a single path, which can be achieved using techniques on *sub-tour elimination*. There are several ways to carry it out. Equation 4 shows Miller & Tucker's [82] formulation. Variable u specifies the visiting order of the vertices ($u_i < u_j$), and $p \ge n$ specifies a maximum number of vertices visited in a TSP.

As mentioned in Section 1.3, the difference between the TSP and the PRP mainly concern typical warehouse obstacles. A significant portion of the work on PRPs focuses on conventional obstacle layouts and how they can be exploited to make optimization computationally efficient [12]. One example is Scholz et al. [83], who build on the IP formulation in Equations 1, 2 and 3 with 77 constraints, many of which are tied to the conventional layout (Figure 12).

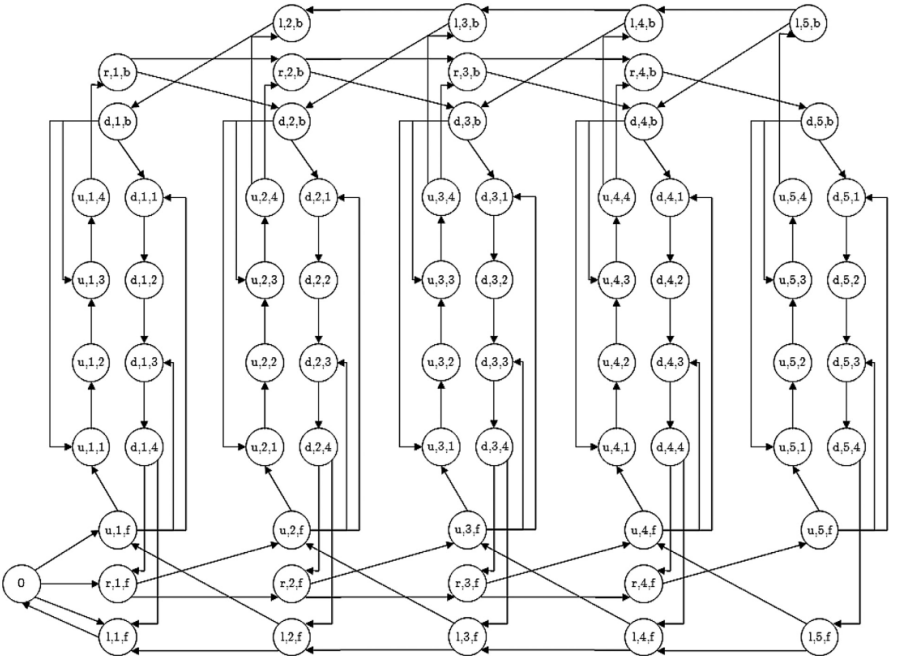

Figure 12: Visualization of how a conventional layout can be used to build a graph for efficient PRP optimization [83].

Mostly, however, PRP formulations for conventional layouts do not use this many constraints or are not as mathematical. Koster et al. [14] and Zunic et al. [45], for example, describe PRP optimization procedures mostly using words.

For the OBP, cost can also be formulated using indexation of edges, as shown in Equation 1. One example is Valle et al. [22]:

$$min \sum_{t \in T} \sum_{(i,j) \in A} d_{ij} x_{tij}. \tag{5}$$

Apart from using a set of edges A, distances are only computed on edges that a single vehicle $t \in T$ (a trolley) traverses. They then add 17 constraints for order-integrity, sub-tour elimination, symmetry-breaking and heuristics tied to the conventional layout. Kulak et al. [16] also formulate cost using Equation 5, but they only use 6 constraints (they do not include the conventional layout explicitly in the formulation).

Another way to formulate the OBP is through set-partitioning. Gademann et al. [41], for example, use the following formulation:

$$min \sum_{s \in S} d_s x_s, \tag{6}$$

where s denotes a batch selected from the set of all feasible batches S. $x_s \in \{0,1\}$ specifies whether a certain batch is selected in the solution and d_s denotes the distance needed to pick the products in that batch, i.e., the distance of a PRP solution. They use the following constraint to ensure that each order is assigned to exactly one batch: $\sum_{s \in S} a_{js} x_s = 1$, for $j = 1, \dots, n$, where a_{js} specifies whether order j is included in batch s and n is the number of orders. Other authors use a similar formulation [5], [42].

For the SLAP, Garfinkel [18] uses the following IP formulation:

$$min \sum_{r \in R} \sum_{z \in Z} n_r c_z y_{rz}\,, \tag{7}$$

where R denotes orders and n_r number of repeats of order r. $z \in Z$ denotes *zones* in the warehouse and c_z the cost of entering a zone, $y_{rz} \in \{0,1\}$ is 1 if a certain zone must be entered to fill a certain order. Note that this formulation does not include any PRP or OBP explicitly. Instead, order-picking cost is computed using the number of times that certain zones in the warehouse are entered by orders. Kübler et al. [24] also provide a crude model of order-picking using a Quadratic Assignment Problem (QAP) (slightly simplified):

$$\sum_{m \in A} \sum_{\substack{n \in A \\ n \neq m}} \sum_{i \in V} \sum_{\substack{j \in V \\ j \neq i}} f_{m,n} d_{i,j} W_{m,i} W_{n,j}, \tag{8}$$

where $m, n \in A$ denote products, $i, j \in V$ denote locations, $f_{m,n}$ pick frequency, $d_{i,j}$ distance and $W_{m,i}$ a binary indicator that is 1 if product m is picked at location i (and respectively for $W_{n,j}$). Similar models can be found within the domain of datamining, where support (frequency) and affinity (distance) between products are used to compute cost [47], [84].

2.4.2 Optimal Solutions

A feasible solution to a PRP, OBP or SLAP is either optimal or sub-optimal. An optimal solution (by some authors it is referred to as an *exact* solution [12]) provides the minimum cost as expressed in Equations 1, 5, 6 and 7. But we also find some disagreement in the literature on the meaning of an optimal solution. Commonly, it is referred to as the solution with minimal distance given that edges between vertices are defined such that they do not intersect obstacles [12], [14], [16], [85]. Other constraints are deemed optional. For example, it is deemed optional to add constraints that restrict the way that pick-rounds are formed because of uni-directional traffic rules, or because certain products cannot be placed below others on a vehicle (Section 2.2). But in real warehouses, these constraints may not be optional, and we may still use optimal algorithms when such constraints are included. Scholz et al. [83], for example, propose optimal algorithms for cases with directional constraints.

In a real warehouse, a PRP, as expressed in Equation 1, can include hundreds of unique vertices (locations), but it is usually much shorter [14]. For common PRPs with a few dozen vertices, the Concorde TSP solver can deliver optimal solutions within a few dozen milliseconds [86]. For 100 vertices, it usually requires around 0.5 – 1 seconds [87]. Concorde uses Linear Programming, Cutting Planes, min-max Duality, PQ trees, Lin-Kernigan heuristics, a Blossom Algorithm and effective ways to achieve sub-tour elimination (it is 130000 lines of C code) [88]. Even though Concorde is publicly available for research, it is used surprisingly seldom in research on PRPs: In the research review by Masae et al. [12], only 13.8% of papers on PRPs propose distance-optimal solutions. Reasons for this low percentage include the attempt to avoid human pickers getting confused when following a distance-optimal pick-round through the warehouse [3]. Furthermore, Masae et al. find that the vast majority of these 13.8% require that the warehouse layout is conventional. Examples of PRP algorithms that provide optimal solutions specifically for the conventional layout include Linear Programming, branch and bound and Dynamic Programming [12], [14], [83]. For optimal PRP optimization on unconventional layouts, there is little previous work:

> it seems that the application of [PRP optimization algorithms] to layouts different to the model containing parallel aisles and a central depot has not been considered at all in the literature [89].

For the OBP and SLAP, optimal solutions are only obtainable within reasonable CPU-time for the smallest problem instances. Briant et al. [15], for example, propose an OBP algorithm that requires between 300-7200 seconds

to generate optimal batches from 20-45 orders. They use a Branch and Price algorithm. A Master Problem (MP) based on a linear relaxation of all possible pick-rounds in an OBP is first formulated. Each possible pick-round serves as a column which can be added to the MP. In each iteration of their algorithm, a Lagrangian lower bound is computed based on the MP. Then, a Restricted Master Problem (RMP) is solved to optimality. The RMP only includes a subset of pick-rounds, which are selected based on a linear relaxation of the PRP. The optimality-proof of the selection of pick-rounds is based on heuristics tied to two conventional warehouse layouts and a cutting plane method. The best pick-rounds are added as columns to the RMP. The procedure continues until the RMP has been solved to optimality (when improving columns cannot be found) or a time-out. Finally, the upper bound is provided by solving the MP with the final set of columns in the RMP. While the value of providing optimality bounds for the OBP is important, Briant et al.'s method only works for conventional layouts and requires significant CPU-time.

For the SLAP, there is not much prior work attempting to find optimal solutions. Boysen & Stephan [64] propose Bounded Dynamic Programming (BDP), a type of Held-Karp algorithm where an intersection between lower and upper bound SLAP solution candidates is formulated. The upper bound is obtained by a greedy heuristic and the lower bound is obtained through a linear relaxation, a greedy heuristic and a local-search Held-Karp algorithm. Due to the exponential complexity of BDP, Boysen & Stephan's experiments are restricted to a conventional layout with up to only four racks. Garfinkel [18] proposes Lagrangian relaxation and various constructive and clustering heuristics to formulate upper and lower bounds for warehouses with a known number of zones. The task is to relocate products such that the number of "multi-zone" orders is minimized. The travel costs are modeled on a zone-level, which significantly simplifies the problem. Even so, to obtain the optimal solution for 40 zones and 100 products, Garfinkel reports that a CPU-time of 3.5 weeks was needed for the experiment.

2.4.3 Sub-optimal Solutions

Common heuristics for distance-sub-optimal PRP optimization include the *S-shape* and *Largest Gap* for conventional layouts [14], [52] (Figure 13). The S-shape algorithm produces an S-shaped path through the warehouse. The Largest – Gap algorithm produces a path which goes around a block of racks and makes incisions into the aisles with pick locations. For warehouses without

conventional layout, optimization using Simulated Annealing and Google OR-tools TSP optimization suite have been proposed [25].

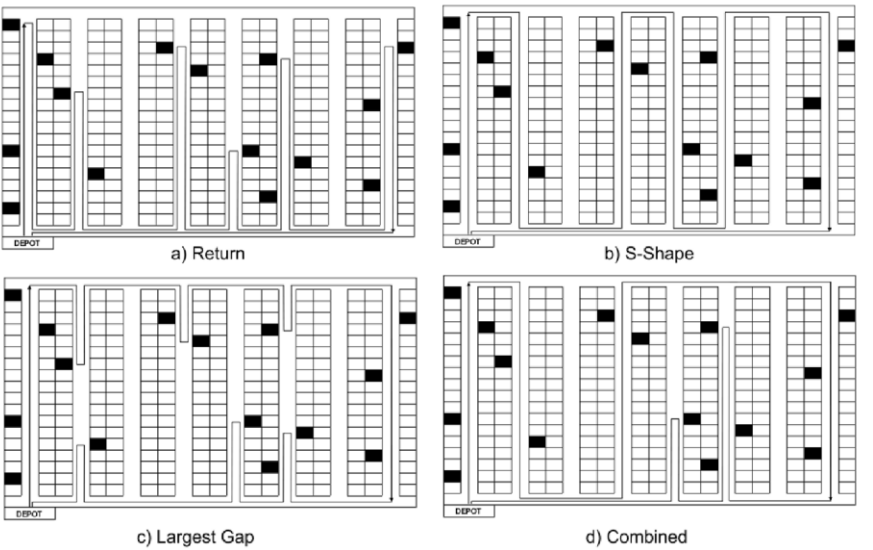

Figure 13: PRP heuristics for a conventional layout (single block) [5]. The black cells denote products to be picked and the lines show proposed pick-rounds.

For the OBP, sub-optimal algorithms can be divided into heuristics and meta-heuristics. Heuristic algorithms can be divided into four categories: Priority-rule based, seed, savings and data mining algorithms [5], [90]. In priority rule-based algorithms, batches are built by ranking and then assigning orders by importance, e.g., using First Come First Serve, First Fit and Best Fit. In savings algorithms, the cost of batches with single orders is first evaluated. This result is then compared against proposals of pairs and triplets of orders in batches. In seed-algorithms, batches are generated in two steps: A seed selection step and a construction step. In the first step, a suitable first order is selected, and orders are then added to it in the second step, until vehicle capacity is exceeded. There are many possible heuristics that can be used for this purpose: Ho et al. [91] propose 11 different heuristics for seed selection and 16 heuristics for construction. Examples of meta-heuristic algorithms for the OBP include Tabu Search [90], Ant Colony Optimization [92], Genetic Algorithms [93] and Variable Neighbourhood Search [94].

For the SLAP, sub-optimal algorithms are also divisible into heuristic and meta-heuristic. As shown in Equations 7 and 8, heuristics can be used to provide a simplified model of the order-picking assumed to be used in the warehouse (compared to modeling it more explicitly using PRPs and/or OBPs). Apart from using the number of times zones are entered (Equation 7) and distances and pick-frequencies between pairs of products (Equation 8), other alternatives include Cube per Order Index (COI) [95] and Order Oriented Slotting (OOS) [62]. COI uses the pick-frequency and volume of a product to compute the ideal proximity to a depot. COI does not include capability for multiple products in an order. OOS, on the other hand, is specifically designed to handle orders with multiple products. Meta-heuristic algorithms for the SLAP include Simulated Annealing [96], Ant Colony Optimization [97] and Evolutionary algorithms [56].

2.4.4 Machine Learning (ML)

There have been attempts at introducing pre-trained ML approaches for the PRP and OBP [98], [99], [100], [101]. They offer to replace decision-time search for optimization of PRP's and OBP's, with pre-trained parameters, which could provide significant improvements in computational efficiency. The parametrization can be achieved by mapping locations, obstacles and PRP or OBP problem instances into a graph where vertices are connected based on relationships defined in the geometric domain. These parameters can then be trained on many sets of annotated problem instances to learn to approximate search policies or solution costs. This approach has both theoretical merits and flaws.

Concerning the merits, it has already been shown that learning based on features in a combined geometric and sequential domain is possible [102]. Search algorithms for games such as Chess and Go, including Monte Carlo Tree Search (MCTS), can be substantially improved by the usage of pre-trained parameters [102]. Strong results have also been achieved on Starcraft 2 [103] and Dota 2 [104].

The relevance of these achievements for PRP, OBP and SLAP optimization can be debated. The Starcraft 2 algorithm (AlphaStar) uses a wide array of game-specific *entities* and a graph of structured actions that can be applied on the entities [105]. It is designed for a specific Real Time Strategy (RTS) game with hidden information. If the environment is turn-based and without hidden information, as in the case of Go and AlphaZero [102], the similarities to the PRP, OBP and SLAP are arguably stronger. One core feature of AlphaZero is

convergence on optimal play, made possible by its self-play training loop [100]. This type of convergence is based on the decomposition of a problem instance into a Markov Chain of states, where the learning of policies to move from one state to the next is assumed to be a differentiable problem. This differentiability is, in its turn, dependent on the gradual building of a training database using experience replay. The objective is stated as generating a policy which beats any other policy's performance on the training database.

There is no immediate theoretical hindrance for why this type of approach could work on problems such as the PRP, OBP and SLAP. The main issue can rather be attributed to more practical reasons: As laid out in Section 2.2, there are many types of obstacle layouts, picking methodologies, constraints and other features in PRPs, OBPs and SLAPs. Realistically, a WMS which includes ML-based optimization modules would also include various heuristic modules [99]. To see why heuristics are needed, the OBP can be used as an example: To get the solution cost of a single OBP candidate solution (i.e., an assignment of orders and visit sequences to vehicles), a set of TSP's need to be optimized. Incorporating learning to approximate TSP costs within an OBP algorithm is naturally complicated by the NP-hard nature of the TSP. Vinyals et al. [106] try learning TSP costs, but they do not offer a competitive result (compared to the Concorde TSP solver, for example). It could still be possible to incorporate a learnt approximation of TSP costs in an OBP optimizer (cost approximation has been shown to be effective in OBPs [107]), but the added burden of development, pre-training and maintenance of such an ML-based optimizer is significant. In the warehouse domain, learning TSP costs is mostly feasible if features, such as rack layouts, are heavily restricted, such as in the work by Seward on conventional layouts [99]. As soon as the TSP is assumed to be generic, the training data needed increases significantly.

There are order-picking problems other than the PRP, OBP and SLAP that may be more suitable for ML. One example is *demand forecasting*, i.e., the prediction of products or orders that are going to be demanded at a future time. This topic has not seen an overly large amount of research within the warehousing domain, but it can draw from a larger body of both practical and theoretic work within the broader logistics and ecommerce domain [108], [109]. Abolghasemi et al. [110], for example, propose ML methods to measure the volatility of demanded products generally, without going into detail on practical applications. Within ML research, there is a variety of methods and datasets which target the same type of problems [111]. Spiliotis et al. [108] apply various ML methods against standard statistical methods in a case-study to predict the demand for 3300 products. A specific difficulty of their dataset

is that certain products occur very sparsely through time. They find that ML methods are promising for this type of task, especially because of their low CPU-time requirements (after pre-training), but that some methods underperform for reasons that they admit are elusive. A well-known problem with ML methods is that they are "black-box" in nature [99], [112]: It is difficult to deduce why they underperform or overperform on a certain task, since their search for parameter combinations is largely autonomous. Hodzic et al. [113], also conduct a demand forecasting experiment to test the difference between an ML method, in the form of a Long Short Term Memory cell (LSTM), against a standard statistical one, in the form of an Adaptive Median. Their dataset consists of 2913 products, and they find that the LSTM is stronger than the Adaptive Median when it comes to forecasting the number of demanded products, but that the Adaptive Median is stronger when it comes to forecasting the specific products that are demanded.

3. Approach

3.1 Feature Selection and Engineering

Before we approach PRP, OBP and SLAP optimization (Section 3.2), we select and engineer a set of features. For each feature, we attempt to make choices that are beneficial for standardization (Section 1.7.1) and the SaaS optimization architecture (Section 1.7.2).

3.1.1 Layout

We propose that standardized order-picking optimization methods need to have capability to handle polygonal obstacles that are distributed in any manner in two dimensions. In our papers, we refer to this scenario as the *unconventional* layout [114]. In reference to related work, we can also call this scenario *the union of conventional, non-conventional and general* layouts (Section 2.2). A significant amount of related work on order-picking optimization is designed exclusively for the conventional layout. But we also note the following:

1. A significant quantity of warehouses do not use conventional layouts.
2. In terms of CPU-time, it is expensive to obtain optimal solutions for OBPs and SLAPs, even for conventional layouts.

For the PRP, there exist algorithms capable of finding optimal solutions for conventional layouts at relatively low CPU-time, including approaches based on linear and dynamic programming (Section 2.4.2). But for the OBP and SLAP, proposed optimal solutions require CPU-times that can be deemed in excess of what is SaaS-suitable (Section 3.4.1). Arguably, potential optimization advantages attained when working exclusively with conventional layouts are exceeded by the advantages attained from generalizing layouts [114].

Bartholdi & Hackman [1] point to an important disadvantage of unconventional layouts that we must address: Added complications in building and applying a digital model of a warehouse. All pairwise distances between the locations, i.e., the distance matrix, need to be pre-computed. If we study conventional layout problem instances, such as "ran-x" or "abc-x" [90], we note that they include entries such as "Aisle 6 Location 22" and "Aisle 13 Location 29", as well as information on aisle widths and lengths. From this information, it is possible to build a distance matrix between the provided locations at high speed. For the unconventional layout without uniform aisles, this is not possible (Section 3.2.1). Consequently, the distance matrix must be built in an onboarding step before a warehouse can start using the optimization service. The distance matrix must also be kept accessible in memory for the optimization service to perform well in terms of CPU-time (Section 3.4.1).

3.1.2 Locations and Zones

We propose a distinction between four types of locations:

1. *Depot/origin/destination locations*: The locations where vehicles start and end pick-rounds.
2. *Obstacle locations*: The outlines of racks or other obstacles.
3. *Product locations*: Each product has one location in the warehouse. In real warehouses, there are cases when a product is stored in multiple locations. This complicates order-picking optimization, since decisions must be made concerning which of the product locations should be visited. In Section 4.1, we exemplify this scenario and show why it is unsuitable for standardization.
4. *Stop locations*: To limit the size of a digital model of a warehouse and to simplify optimization, many product locations can be set to share a single "stop location". For example, if we have 20 products within an area of 4 m^2, we can use a single stop location, instead of 20 different product locations.

Regarding stop locations, in Section 2.2 we stated that surjective relationships between many products and fewer locations (in specific areas) are similar to *zones* in a warehouse. We can see the similarity if we think of a precision or granularity regarding the digitization of product locations. We define a stop location using a fine granularity level, claiming that movements within a few square meters do not have to be accounted for when finding solutions to order-

picking problems. This can help reduce memory and computational requirements (Section 3.2.1). Order-picking optimization is carried out on the stop-location level, so the granularity with which stop locations are defined dictates the precision in optimization: The larger area that we assign to a single stop location, the lower the achievable precision. In some scenarios, it may be beneficial to work with multiple levels of surjective products to locations relationships. For SLAP optimization, for example, we could start with a broader area for product-location assignment and move gradually toward assignment of more exact locations.

Arguably, the *stop-location* term generalizes the *zone* term. We could use the zone term instead, but the zone term is usually associated with a coarse granularity level, defined in terms of larger areas.

3.1.3 Depots

For the PRP, we assume that both the origin and destination can be any location in the warehouse. We consider requirements of a single depot as strict, since PRP optimization can be set up for multiple depots with relative ease using "dummy nodes" [115]. Since we include the PRP in our work on the OBP and SLAP, we do not regard single-depot configurations as necessary for these either. SLAP optimization involving the Quadratic Assignment Problem (QAP) or pick-frequency heatmaps [25] assume a single depot, but could potentially be modified for the multi-depot case.

3.1.4 Dynamicity and Stochasticity

Standards are lacking on how to define dynamicity in PRPs, OBPs and SLAPs. For the sake of standardization, we propose that work on static, rather than dynamic versions of these problems, has precedence. The focus is on the optimization quality that can be achieved given all the data available in a static PRP, OBP or SLAP instance. There are many choices regarding how to define problem instances, even in static settings (Section 2.3).

We only consider stochasticity with regard to instance generation (Section 2.2). We propose pre-generated deterministic instances, following the same general structures as used in the TSPLIB [70] and Cook [71] instances (Section 2.3.1).

3.1.5 Order-integrity

We use order-integrity in our work on the OBP. It follows that a single order can always fit on a single vehicle in the warehouse. We assume that we are not aware of whether an order has been split at an earlier stage (by a Warehouse Management System (WMS), for example) to make this possible. In certain implementations of order-picking, e.g., single-product picking robots or wave-picking, order-integrity may not be beneficial for operations. But it is beneficial from a standardization perspective, as single-product picking or wave-picking can be modified into OBPs using relatively simple pre-processing: Single products can be redefined as orders with single products.

3.1.6 Product Constraints and Traffic Rules

Warehouses often contain products that are hazardous, in need of cooling or specialized placement on a vehicle. We do not consider these types of specialized products for standardized versions of the PRP, OBP and SLAP.

We work with scenarios where travel costs are assumed equivalent between pairs of locations. If the warehouse uses uni-directional travel rules in aisles or cross aisles, they can be imposed using digitization techniques that do not impact optimization CPU-time (Section 4.5).

3.1.7 Capacity Constraints

Capacity constraints have a significant impact on optimization performance, but they are challenging to standardize due to the high variability of warehouse vehicles and picking methods. Possible capacities for vehicles include number of orders, number of products, xyz dimensions and carry weight. For the SLAP, capacities can also be defined in terms of number of products or xyz dimensions for locations. We mainly work with number of orders and/or products in our experiments on PRPs, OBPs and SLAPs.

3.1.8 Cost Function

There are several choices for cost functions for the PRP, OBP and SLAP, including distance, time and profitability (Section 2.2). We use distance in our experiments. From the standardization perspective, distance is beneficial since it is an unambiguous metric, assuming it is modeled such that it accurately represents the geometry of a warehouse. Travel distance is often chosen due to

its close correlation with travel time [3]. But time may also include components that are difficult to standardize, e.g., search-times for products, administration and unexpected delay bottlenecks. Worth noting is that the cost function will always be delimited to some extent: The warehouse is not the only part in the logistics chain, and profitability may be affected by factors that are difficult to optimize [116].

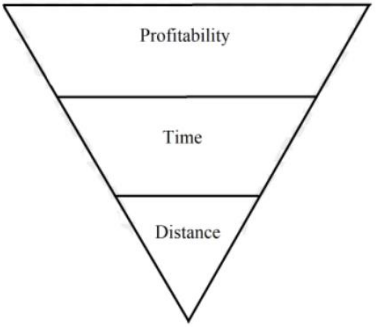

Figure 14: The amount of data needed to work with profitability, time and distance cost functions [100]. Profitability may be of higher interest to managers, but that does not necessarily mean that it is a better cost function from a standardization perspective or optimization.

Hence, distance is benefited by its simplicity in terms of both implementation and interpretability. For the SaaS business (Section 1.7.2), it makes sense to carry out an initial Proof of Concept (POC) for WMS customers in a way that is as explainable as possible. Work with more complex KPI's should be conducted after work with more basic KPI's has been completed.

We choose the Euclidean norm (Section 2.1) for our computations of shortest paths and distances. For conventional layouts, heuristic models for distance can be used (such as counting the number of aisles that are entered) [58]. For the unconventional layout, Euclidean distance is arguably a stronger choice than the two alternatives proposed by Gibson & Sharp [58], i.e., Manhattan and Chebyshev. The main advantage of Euclidean paths and distances are their stronger descriptive properties. In Figure 15, we suggest that it is easier to draw conclusions regarding the quality of a Euclidean-based PRP solution, than a Manhattan alternative. This can be useful if we wish to visually validate the digitization process (Section 3.2.1). After a warehouse graph has been built, a few mock PRPs are generated and solved using the Concorde TSP solver, and these PRPs are then visually inspected for correctness.

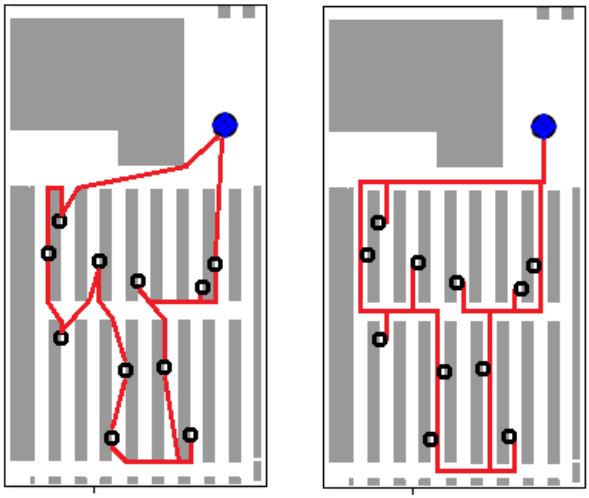

Figure 15: The same PRP solution using Euclidean (left) and Manhattan (right) paths/distances. In the Euclidean case, we can see that the path looks close-to-optimal. In the Manhattan case, it is less clear.

3.2 Optimization

3.2.1 Digitization and Preprocessing

As we discussed in Section 3.1.1, our assumption of an *unconventional* layout requires the building and storing of a digital model of the warehouse. For this purpose, we use a warehouse digitization process introduced in 2019 and 2021 [114], [117].

A warehouse can be modeled as a graph $G = (V, E)$, consisting of vertices V representing locations in the warehouse, and edges E, which represent possible paths to move between the vertices. Janse van Rensburg [117], proposes a method in which all shortest paths and distances between the vertices is computed such that obstacles are circumvented. The method can be summarized as follows:

1. A 2D top-view image of the warehouse is used as reference.
2. Stop locations are added as x, y coordinates and corresponding mappings between product locations to stop locations are generated.

3. Obstacles of any 2D shape and rasterized edges (drawing of lines on a raster) between locations are plotted/generated.
4. Rasterized edges that include cells located within an obstacle are removed.
5. Graph G is constructed from the locations (represented as vertices) and the remaining edges.
6. The Floyd-Warshall algorithm is used to compute all shortest walks and distances between vertices.

Experiments show CPU-time and memory requirements to build various sizes of G, shortest walks and distances [114]. The largest graph includes 6491 vertices and 306 obstacles and requires 1.2 GB of memory and ~18 hours CPU-time. Most warehouse graphs can be constructed using less than 100 MB, however. Also note that the number of locations can be significantly reduced if we accept a lower precision of the optimization results (Section 3.1.2).

3.2.2 Picker Routing Problem (PRP) Optimization

For PRP optimization, three sources of TSP optimization software are used: The Concorde TSP solver (Section 2.4.2), the OR-tools TSP optimization suite (Section 2.4.3) and Simulated Annealing [119].

A Linux-compatible wrapper to Concorde is available online [86] and it is also the version used in all our implementations. The ability of Concorde to produce optimal solutions to the TSP is well-documented, but its ability to produce optimal solutions to the PRP is not. On the one hand, no attempt to prove Concorde's optimality on PRPs is carried out in this project. On the other, no substantial evidence is identified which indicates that it performs sub-optimally. When Concorde's performance on PRPs is compared to its performance on TSPs, no substantial difference is observed [25]. From visual inspection of hundreds of solved PRPs and TSPs, sub-optimal looking solutions are not observed. A counterargument is that it is difficult to know what an optimal solution looks like, especially when working with PRPs that are constrained in various ways. For example, we could have unidirectional travel rules or multiple depots that creates challenges for visual inspection. We also note that the use of surjective relationships between products and locations (Section 3.2.1) could have a detrimental impact on Concorde's performance.

We continue with the CPU-time that Concorde uses. Commonly, PRPs are shorter than ~50 locations, and Concorde delivers solutions to these in

fractions of a second (but with a high variance [87]). Concorde may be fast enough for PRP optimization in isolation, but it is less clear that it is fast enough for use within OBP and SLAP optimization. On the one hand, we can easily motivate using Concorde within OBP and SLAP optimization: It provides a model for distance-optimal pick-rounds through the warehouse. But on the other, the number of PRPs that Concorde needs to solve in OBPs and SLAPs, rises exponentially: Assuming Concorde requires 10ms to solve a single PRP, it requires 1 minute to solve 6000 PRPs. Even in relatively small OBP and SLAP instances, there are orders of magnitude more possible PRPs. As an example, an OBP where 30 orders should be distributed into 5 batches, may have around 10^{17} possible PRPs (a rough estimate provided by the Stirling number).

To reduce CPU-time, we also investigate usage of distance-sub-optimal PRP optimization within OBP and SLAP optimization [25], [120]. A joint conclusion from this work is that the combined use of Concorde and sub-optimal PRP optimization is motivated, but that the large variability in search space and features makes it is difficult to provide any concrete evidence for how much one should be used over the other. One could, for example, hypothesize that the utility of sub-optimal PRP optimization grows with instance size: We do not carry out experiments of a large-enough scale to thoroughly test this, however.

For sub-optimal PRP optimization, we primarily use the Google OR-tools TSP optimization suite [118]. It includes at least five different TSP optimizers, and per default it selects one automatically. One disadvantage of the OR-tools TSP optimization suite is that some of its functionality is not extensively documented. For example, setting it to use “Simulated Annealing” often leads to worse performance compared to letting it choose optimizer automatically, while it is not clear why this is the case, nor explained how it does the automatic selection. Another example is a lack of information regarding how to control the CPU-time used for TSP optimization. This is crucial information when deciding on how to trade-off CPU-time against distance minimization, which is especially relevant if one seeks to use OR-tools as a cost approximator for TSP solutions produced by Concorde (more on this in Section 3.2.4).

For sub-optimal PRP optimization, we also use a publicly available Simulated Annealing optimizer [119]. This implementation is slower, in terms of CPU-time, but simpler compared to OR-tools, and we mainly use it for certain visualization scenarios.

In certain industrial OBP and SLAP scenarios, a case against distance-optimal PRP solutions can be made. Distance is often not the KPI that a WMS operation uses, and the industrial setting (Section 1.7.2) is a SaaS that sells services to a WMS. In other words, the SaaS needs to be mouldable to whatever KPI the WMS uses. If a WMS provider does not use distances in their KPI model, the SaaS provider should not require it to work with them. If the WMS provider uses a more basic cost function to evaluate performance, such as SMD in the case of the OBP, or a support-distance dot product in the case of the SLAP, the SaaS provider can avoid PRP optimization altogether. PRP optimization on a SLAP picking-log using Concorde can be thousands of times slower than computing a support-distance dot product [25]. Hence, if the WMS uses an evaluation based on a support-distance dot product, it makes sense for the SaaS provider to also use it until the customer demands something else.

3.2.3 Order Batching Problem (OBP) Optimization

The set-partitioning formulation by Gademann [41] (Section 2.4.1) is used as foundation for all our work on the OBP. One difference is the addition of a set of vehicles:

$$min \sum_{b \in \mathcal{B}} D(b) x_{mb}, m \in \mathcal{M}, \tag{9}$$

where $b \in \mathcal{B}$ is a batch generated out of the set of all valid batches, $D(b)$ is the distance of b as provided by PRP optimization, $m \in \mathcal{M}$ is a vehicle selected from available vehicles and x_{mb} a binary variable which is 1 if m is assigned to pick b. We then add constraints for order-integrity, minimum required location visits and vehicle capacity [114]. Order-integrity, precedence and capacity constraints reduce the number of possible batches and thereby the number of possible PRPs. In a pallet picking usecase with all these constraints (Section 4.3), the number of possible combinations of pallets on a fork-lift is so small that optimization often cannot yield significant savings. In other cases, the number of possible PRPs can be very large. In related work, we find that optimal OBP optimization suffers from being tailored for specific scenarios (Section 2.4.2). To model an OBP as a convex optimization problem, for example, many assumptions must be made with regard to the features (such as layout and capacity constraints). From the perspective of a SaaS - business (Section 1.7.2), another issue with optimal optimization is that it requires a significant amount of CPU-time. For a customer driven SaaS, it is advantageous if optimization results can be delivered quickly (Section 3.4.1).

With this reasoning, we design and provide sub-optimal optimizers for the OBP [87], [120] (Papers 2 and 3). They are based on heuristics and meta-heuristics. The main optimizer is called Single Batch Iterated (SBI) and it uses constructive heuristics in a Seed algorithm (Section 2.4.3). Orders are added to a batch semi-randomly based on a Sequential Minimal Distance (SMD) heuristic, which is used to compute (and later query) distance estimates between orders. Cost evaluation for a candidate batch is achieved using optimal or sub-optimal PRP optimization. The procedure continues with semi-randomly generated and evaluated batches until convergence or a pre-designated timeout. SBI is evaluated in terms of computational efficiency against optimal OBP results on smaller existing conventional layout instances [120]: An average gap to optimality of 2.3% is reported against Briant et al.'s [15] Branch and Price method. Importantly, the proposed sub-optimal results require significantly lower CPU-times compared to Briant et al. For larger instances where optimal solutions are not available, we find that solution improvement often decreases to 1% per minute after 30-60 seconds of optimization. For example, an initial SBI solution may take 5-10 seconds to obtain. After 5 minutes of additional optimization, the initial solution is only improved on by 4-7% [120].

The main meta-heuristic optimizer is called Metropolis Batch Sampling (MBS). First, an initial solution is generated using the seed and SMD heuristics. Then, batches are altered iteratively using the Metropolis algorithm by swapping orders between batches and computing the PRP costs of new batches. The Metropolis acceptance ratio controls whether new batches and their corresponding costs are accepted or rejected.

In terms of performance evaluation, there are advantages and disadvantages to both SBI and MBS. In terms of computational efficiency, SBI performs better on OBP's, but it is designed specifically for that problem, relying heavily on the SMD heuristic. MBS is more easily adopted to optimize other problems, such as the SLAP, where distance estimates between orders are not necessarily as directly relevant. On the OBP experiments conducted, the relative strength of SBI can be explained through two factors: 1. The predictive strength of the SMD heuristic. 2. The relatively low CPU-time needed to compute SMD, as well as the temporary storage of SMD values in memory. As is often the case in the engineering of optimization algorithms, multiple improvements to both SBI and MBS are developed during the project. For SBI, improvements mainly regard computational efficiency through reuse of certain computations. For MBS, improvements include the addition of temperature (i.e., Simulated

Annealing), seed selection strategies and more effective usage of cost approximation.

3.2.4 Storage Location Assignment Problem (SLAP) Optimization

Our proposed optimization models for the SLAP build on our work on PRP and OBP optimization (Section 3.2.2, 3.2.3). We propose two versions of the SLAP: The OBP-based SLAP and the TSP-based SLAP[2]. There are two main differences:

1. *Cost evaluation:* In the OBP-based SLAP, solution cost is obtained by optimizing an OBP. In the TSP-based SLAP, solution cost is obtained by optimizing multiple PRPs (TSPs).
2. *SLAP features:* In the OBP-based SLAP, products that are newly arrived in the warehouse are assigned a location. In the TSP-based SLAP, the locations of products already in the warehouse are swapped.

Concerning cost evaluation, it is assumed that a SLAP problem-instance includes a future-forecasted *picking-log* with information on products that need to be picked in the warehouse. In the OBP-based SLAP, the picking-log consists of orders (that need to be batched in an OBP), and in the TSP-based SLAP, the picking-log consists of pick-rounds (which may have been generated using OBP optimization at an earlier stage). Some researchers, such as Kübler et al. [24], also include the forecasting of the picking-log (demand forecasting) as part of their SLAP optimization model, but we argue that standardization suffers if this is required. The core question the SLAP asks is where products should be stored, not to predict the number of products that will be picked. An issue with demand forecasting is that it is dynamic in nature, as it is based on seasonal trends. Such trends are difficult to standardize due to the high variability of products.

Building on our work on OBP optimization (Section 3.2.3), our SLAP optimizers rely on heuristics and meta-heuristics. The OBP-based SLAP optimizer uses a Nested Metropolis algorithm and is divided into four layers: The first (outer) layer runs a Metropolis algorithm, the second layer approximates the cost of a SLAP sample using a model based on the Quadratic Assignment Problem (QAP). The third layer computes ground truth OBP costs using the SBI optimizer for samples which have passed an accept filter based

[2] See Section 1.5 for a rationalization for using "TSP" in the title.

on the QAP estimates. The fourth layer is comprised of PRP optimization as used by SBI. See Paper 4 [107] for a diagram of the optimizer.

One key problem with the OBP-based SLAP is that solution quality of a single SLAP candidate sample (an assignment of products to locations) requires an OBP to be optimized. An important issue is that there is no proposal with which to efficiently solve realistically sized OBP instances to optimality (Section 2.4.2). The OBP-optimizer used (SBI) produces sub-optimal solutions. We could argue that usage of the OBP to obtain SLAP solution costs is a reasonable choice. After all, if a warehouse uses order-batching, it makes sense to compute SLAP solution quality using OBP-optimization, even if it the costs are sub-optimal. Nevertheless, the usage of sub-optimal results to guide a Metropolis algorithm (for SLAP optimization) can be problematic for two reasons:

1. *Stability issues.* Building on a line of argumentation by Mantel et al. [62], an inaccurately evaluated solution candidate will cause a Markov chain to jump around less ideal search regions. Furthermore, we use two levels of approximation, since we approximate OBP costs provided by SBI using a Quadratic Assignment Problem (QAP) model. While QAP costs are fast to compute, they are not necessarily proportional *enough* to OBP costs. Costs in a QAP are provided by the sum of all pairwise distances between a set of product locations, multiplied by their pick-frequency. Experiment results show that usage of the QAP approximations can be beneficial for optimization of the overall SLAP (at least certain versions) [107], but that its generalizable properties can be debated.
2. *Hyperparameter search space*. A combination of hyperparameters in both the OBP and SLAP optimizers needs to be found. For the OBP optimizer, these hyperparameters are N (number of samples), C and P (floats controlling the amount of feature-distance between samples). For the SLAP optimizer they are N (number of samples), λ (float controlling the amount of distance between samples) and φ (choice of relevance values). The search of a strong combination of these hyperparameters is possible, but it requires a significant amount of quantitative experimentation. If it takes 4 seconds to obtain an OBP solution within 10% of optimality (for a "medium" – size instance) [120], it takes 8 hours to generate 7200 samples. Since the SLAP search space for but the smallest instances is many orders of magnitudes higher, 7200 samples is inadequate to guarantee convergence on a strong solution.

From a standardization perspective, we could question whether in-depth quantitative experiments are suitable for the OBP-based SLAP. Arguably, feature standardization has precedence. Besides the listed features in Section 2.2, there is little agreement on how to standardize SLAP-specific features, such as the reassignment distance (the cost needed to swap locations between products). For the OBP-based SLAP, the easiest reassignment scenario is used, namely the one where new products appear in the warehouse and reassignment distance can be set to zero (since the location assignment of new products is mandatory). This is clearly a simplification of a realistic scenario, where warehouse managers also want to know when locations between products already in the warehouse should be exchanged.

The optimization algorithm for the TSP-based SLAP [25] (Paper 5) is similar to the one above, but Nested Metropolis sampling is replaced with Nested Annealing (a temperature parameter is added). It excludes the OBP, and instead, all the order-batching forecasting is assumed to have been carried out at a prior stage in the overall WMS. While Paper 5 improves the computational efficiency of the MCMC algorithm by usage of restart heuristics and sub-optimal TSP optimization, the main focus is on the modeling of a more standardized reassignment scenario: If new products are predicted to arrive, they are assigned uniform random free locations in a baseline sample. If they are already in the warehouse, the baseline sample keeps them at their current locations. The picking-log is then passed on to the SLAP optimizer which finds suitable location swaps between products, considering both optimal distances of TSP solutions for the picking-log, as well as sub-optimal reassignment distances.

We report cost savings around 23% after 1 hour of optimization of the OBP-based SLAP and cost savings around 30% after 8 hours of optimization of the TSP-based SLAP. The reassignment distance in the TSP-based SLAP may pose a more serious problem than the sub-optimal estimates in the OBP-based SLAP. The issue with the former is that minimization of reassignment distance is not easily achieved alongside minimization of picking-log distance. To minimize the latter, many products need to exchange locations with other products. But the reassignment distance is positively correlated with the number of location exchanges [25].

3.3 Benchmarking

PRP, OBP and SLAP features should be described in instances such that they are as representable as possible, while allowing for simple experiment reproducibility. Currently, there is little agreement regarding a benchmark standard for the PRP, OBP and SLAP.

The representation of a warehouse's layout is one point of contention. In instances designed for the conventional layout, such as Foodmart [22], HappyChic [69], ran1-3 and abc1-3 [121], features such as "number of aisles" and "intra-aisle-distance" are used to describe the layout. There are also instance file formats for the conventional layout that include a distance matrix. For the layout-agnostic case, neither of these options are appealing. Features, such as number of aisles and intra-aisle-distance, cannot be used, and the distance matrix takes up a lot of space in the instance, while still not providing enough information to allow a recreation of the warehouse layout, nor of exact picking paths through the warehouse.

We propose a different approach, but it also has caveats. In our generated instances [25], [87], [107], [120], we leave out the distance matrix, and instead provide the coordinates of all locations, including the ones for polygonal obstacles. The distance matrix must then be generated in such a way that no path between two locations intersect any obstacle. The main problem with this approach is that the distance matrix must be reproduced before minimized distances can be checked for correctness. We still argue against including the distance matrix directly in the instance, since the distance matrix is only one of several files needed to properly validate an experiment. To validate an experiment, the distance matrix itself needs to be validated, and for that, only two options are possible: Either to provide the whole graph of shortest paths in the instance, or to provide the data needed to generate the graph. The shortest paths and distances can take up as much as 50 megabytes of memory for a warehouse with 500 locations [114], which is arguably excessive for benchmark data.

3.4 Industrial context

3.4.1 SaaS CPU-time and Deployment Options

A key issue is the time it takes between an optimization request being sent from the WMS customer and receival of the response from the SaaS. In this section, we discuss why this topic is important and how SaaS CPU-time can be minimized.

Since results to common PRPs in warehouses can be obtained in fractions of a second, it makes sense to offer a PRP optimization service as "instantaneous" and/or even "optimal" (using the Concorde TSP solver with a license, for example). The customer sends a request for PRP optimization and "instantaneously" gets a response with the shortest possible pick-round. In order to achieve this, we must assume that the optimization server has the digital files of a warehouse stored in memory (Section 3.1.1). If we assume that we wish to offer the same service for all warehouses, we should assume that there will be warehouses whose digitization files are going to take up significant memory. Since it takes time to load these files from disk into Random Access Memory (RAM) such that they can be used by the optimizer, we need to keep these files highly accessible. The most significant file in this regard is the distance matrix, i.e., the file which provides all pairwise distances between locations in a warehouse. If we assume that we want to store the distance between two locations in 16 bits, we need $1.6 * 10^9$ bits (200 MB) for a distance matrix with 10000 locations (we do not wish to half it since we may want to include capability for asymmetrical distances). While we can reduce the number of locations (Section 3.2.1), we can also assume that we need memory for other types of data. For PRP visualization (Section 4.4), for example, we need to have shortest walks between the product visits in the PRP, and these take up more memory than the distance matrix [114]. Assuming that the digital files for a warehouse take up 1 GB, it can take a few seconds to load them into RAM. Following this reasoning, we ask two questions:

1. Is a WMS customer willing to wait a few seconds for a response to a basic optimization request? For example, we can imagine a scenario when the customer sends a PRP request with five pick locations to test the service for the first time.
2. What are the loading speeds and costs from storage for different SaaS deployment options, like Infrastructure, Platform and Function as a Service (IaaS, PaaS and FaaS)?

Regarding the first question, the WMS customer can be assumed to already use PRP optimization in some form (e.g., if the warehouse uses a conventional layout, it may use simple heuristics like the *S-shape* algorithm). It may not be as good as the Concorde TSP solver in terms of distance minimization, but it is likely very fast. To ensure customer satisfaction, we suggest that the files needed for optimization by the SaaS should be pre-loaded into RAM.

Regarding the second question, the files can be kept accessible in RAM if we deploy a server using IaaS or PaaS (Section 2.3.2). For a FaaS-based solution, the files need to be loaded upon request receival (cold start). Due to complexities involved with this latter option, the relative cheapness of keeping one or a few servers idling (at least for a small-scale SaaS business) we suggest a non-FaaS option. Deployment pricing is a complex topic, however, and FaaS may be a preferable choice with only small adjustments to the cloud business operation. When it comes to selecting between IaaS and PaaS, we do not regard either as superior. An IaaS may have more bare-metal options compared to a PaaS, but assuming we need load-balancing and auto-scaling, among other customizable tools, the final system can end up looking similar following both alternatives.

For OBP and SLAP optimization, "instantaneous" CPU-time is often unfeasible. As we have discussed in Section 2.4 and 3.2.3, optimization of realistic OBP and SLAP instances require significant CPU-time to reach competitive cost savings. In our industrial work, however, we observe that warehouse managers often push for optimization responses to be delivered as fast as possible. Because of this, we include a "computational_time" parameter in the OBP and SLAP request APIs, where the customer can specify the maximum allotted CPU-time for the optimization request. If the customer does not use this parameter, the default option is set to "minimal", i.e., the service assumes that the request should be optimized as fast as possible. We inform customers about the relationship between optimization savings and CPU-time, but we often find that they prefer this "minimal" option. The reason for this can partly be attributed to integration challenges (Section 3.4.2).

3.4.2 SaaS – WMS Integration Challenges

From the SaaS-business perspective, it is relatively easy to explain to a potential WMS customer how PRP optimization can be integrated into their existing system: After the WMS has constructed a pick-round, they translate it to the format of an optimization request, send it to the SaaS, and within a pre-designated timeout (e.g., 1 second), they obtain a response with an optimized

pick-round. If a response is not received before the timeout, the WMS proceeds to use the pick-round they have from earlier.

For OBP optimization, it is more difficult to define such timeouts. A key component in OBP optimization is the computation of distance estimates between input orders (e.g., using SMD). But since the number and contents of input orders may vary widely between different warehouses, it is difficult for the SaaS to predict the CPU-time that may be required for this type of distance estimation. Furthermore, there are variations of OBP request-response set ups that the SaaS needs to have capability for, such as:

1. *Single batch*: The WMS sends a set of orders to the SaaS and receives a single batch in response, together with the orders that are excluded from the batch.
2. *Multi batch*: The WMS sends a set of orders to the SaaS and receives a set of batches in the response.

The WMS may need to implement *polling* as a way to integrate request-response cycles robustly: In the polling set up, the WMS first sends a request and then polls the SaaS until a response is ready for collection. Additionally, it is clearly more complex for the WMS to post-process an OBP optimization response than a PRP response: The WMS should (ideally) check that the SaaS-proposed batches are valid: Are batches within vehicle capacity limits, are included/excluded orders duplicated etc. Besides such tests on the WMS end, the batches may need to go through additional software to be translated into pick-rounds.

For SLAP optimization, integration is more difficult. Firstly, it can generally be assumed that CPU-times are going to be high enough to necessitate polling or a similar solution. Secondly, the WMS needs to share a *picking-log* (Section 3.2.4), i.e., order-picking data that will be used for optimization cost estimates. WMS's usually store such data, but extracting and sharing it can be challenging. Thirdly, it needs to share the products that should be assigned or re-assigned a location in the SLAP request, as well as available (empty) locations (if relevant). This type of sub-selection is necessary as requests cannot be arbitrarily large for a reasonable SLAP optimization set up. WMS customers may be reluctant to implement this sub-selection, however.

As a SaaS provider, we often find that negotiations with WMS customers get stranded due to these types of integration challenges. The challenges for OBP and SLAP optimization described above may be regarded as technically surmountable on our SaaS end. For example, the initial work on OBP-based

SLAP optimization was deemed too complex for integration and therefore we commenced work on the TSP-based SLAP. There are often several additional challenges on the WMS end that we do not see, however. WMS's are complex and often imperfect, with brittle dependencies and limitations on what can be achieved. Oftentimes, the WMS providers are on a tight budget and cannot allocate a sufficient number of man-hours for SaaS integration. Sometimes, a WMS manager enthusiastically starts to integrate OBP and SLAP optimization, only to give up after finding problems that they themselves were not initially aware of.

3.4.3 Number of Warehouses per Cloud Container

Kairos Logic AB uses a custom version of the Google Cloud Platform's (GCP) Appengine Flexible. It uses a docker image to build PaaS containers. The image includes a URL that points to a cloud bucket that holds warehouse files (distance matrix etc.). When a new container is launched, it starts by downloading the files from the URL and loads them into RAM. An important question is the number of warehouses that should be included in the bucket that the URL points to. We explore two possibilities:

1. The URL points to a bucket with the files for a single warehouse. The container instance is dedicated to a single warehouse.
2. The URL points to a bucket with the files for all warehouses. The container instance is deployed with enough memory to include all warehouses.

The first option is more scalable from a technical standpoint, since we assume that there will always be a container capable of storing all the relevant files for a single warehouse. It is more expensive cost-wise, however, since the minimal number of container instances is going to be equal to the number of warehouses. If no requests come in from a specific warehouse, there will still be a dedicated server running for it.

The second option is cheaper, since it allows the minimal number of instances to be 1 (Appengine Flexible minimum). On the other hand, it clearly comes with scalability issues, as the amount of RAM for a single container instance is limited.

4. Additional Projects

4.1 Products with multiple locations

It is common that warehouses store the same product in multiple locations. Usually, these locations are close to one another and used to reduce the risk of a product running out in a single location. For order-picking optimization, this constitutes a problem, as we need to choose which location a vehicle should visit to pick a product. Furthermore, the vehicle might need to visit more than one location for the product quantity in the order to be filled. It is difficult to motivate the inclusion of this scenario in standardized PRP, OBP or SLAP optimization. The scenario is intertwined with product put-away and replenishment, which are not part of the main scope of the dissertation (Section 1.1).

4.2 Batching based on truck loading precedence

The OBP model in this dissertation (Section 3.2.3) does not delve into what happens with orders after they have been picked and delivered to the depot(s). But in real warehouses, the OBP is just one step in a logistics chain. One of Kairos Logic AB's clients asked whether OBP optimization can be extended to encompass the next step in their logistics chain. A portion of one of their warehouses has a staging area, where picked batches are placed before they are loaded into delivery trucks. After loading, the delivery trucks are sent on an outdoor path with *delivery points*, where the batch closest to the back door is unloaded at the first delivery point, followed by a batch further inside for the second delivery point and so on. The batches should be placed in the staging area such that the one closest to the delivery truck is the one to be unloaded at the last delivery point. This set up comes with certain implications: Firstly, two types of batches can be identified. The first type is the *warehouse vehicle* batch, which is a collection of orders which are close to each other in the warehouse. The second type is the *delivery truck* batch, which is one per delivery point. This second batch may be much larger than the first, since a delivery point may contain more orders than can fit on a single warehouse vehicle.

The approach chosen for this problem is to optimize an OBP for every delivery point. The first OBP is for the last delivery point and if it has 50 orders, then an OBP is constructed with these 50 orders. There are no flat walls inside the delivery truck to separate the delivery point batches, as the orders are instead stacked more efficiently using a 3D knapsack problem optimizer. Ideas to integrate this knapsack optimizer with the OBP proved overly complex for the implementation. The placement of the batches before the delivery truck loading is therefore only a rough estimate of how the orders are subsequently placed inside the truck.

4.3 Pallet stacking and safety

The capacity constraints of a warehouse vehicle can take various forms. For the OBP experiments in this dissertation, capacity is often described in number of orders. This is adequate when the vehicle carries bins, one for each order, or when orders are first prepared as empty shipment boxes that are placed on the warehouse vehicle. These bins or boxes are filled with the products belonging to the respective order as the vehicle moves around the warehouse.

It is also common that orders are fully packed pallets that are loaded by a forklift at the pick locations. The stacking of pallets is constrained in various ways. In one project, the aim is to stack as many pallets as possible on a forklift, where the stacking depends on the carrying capacity of a pallet, i.e., the weight that can be placed on top of a pallet, as well as their length, width and height dimensions and the length of the fork. Usually, the dimensions permit a single tower of pallets to be placed on the fork, but if the width of these pallets is low enough, a second tower can be placed in front of the first.

Quantitatively, this type of pallet stacking is relatively trivial. The number of possible stacks is usually not that large (normally, 1-4 pallets are placed on the forklift) and oftentimes the stack is made up of several pallets picked from the same location. There are also caveats with targeting this type of problem for quantitative optimization.

Firstly, pushing the bounds for number of pallets on a forklift poses a safety hazard. In one project, the safety of the stacking was originally judged by experienced human pickers. But management wanted to improve efficiency and informed them that they should stack pallets as proposed by optimization software. As is often the case with software, however, it can only partially replace the judgement of experienced human pickers. One conclusion of this project is that the pickers tend to care less about safety when a software dictates

what should be placed on the fork and how. Subsequently, an incident with a broken pallet ensued. The cause was a forklift that was improperly stacked because of a misunderstanding between warehouse management and Kairos Logic.

Another, less critical caveat with this type of optimization is that the forklift needs to unpack the stack every time a new pallet is to be added to it. For example, if a forklift carries three pallets when it arrives at a new pick location, it needs to unload them from the fork, before loading them again with the new pallet. Distance minimization becomes more questionable as a KPI for an implementation of this procedure. Apart from the time needed to unpack and pack the stack of pallets, the picking path through the warehouse can rarely be the shortest one: If the forklift is set to pick the bottom and inner-most pallet first, followed by one on top or in front of it etc. adhering to constraints, the picking path is decided by where the pallets are located, rather than by PRP optimization. Some alternatives are possible, where the stack gets re-arranged at some pick locations. While there is no doubt that all these considerations can be included in an optimization model, it is questionable whether it can be standardized for multiple types of forklifts and pallet types.

4.4 Graphical User Interface (GUI)

For a company or research project engaged with PRP, OBP and SLAP optimization, it is important to be able to visualize order-picking before and after optimization. It is important both as a debugging tool, to ensure correctness before and after optimization, and to convey a message for potential investors, customers and researchers. The main point of the visualization is to present the workings of the optimization process in a pedagogic and intuitive way. Generally, the difficulty in creating good visualizations follows the complexity of the problem: PRP optimization is easier to visualize than OBP optimization and OBP optimization is easier to visualize than SLAP optimization. The main idea of PRP optimization can be visualized by showing a single or a few pick-rounds before and after optimization. OBP optimization requires the showing of one pick-round per batch, so one can see that the products in a batch are relatively close to each other. SLAP optimization requires the showing both of multiple pick-rounds, as well as a path or multiple paths related to the reassignment penalty.

The visualizations thus tend to include multiple pick-rounds, even for small problem instances, and if these are plotted together on a single picture, it becomes cluttered and difficult to understand. One fix for this problem is to

use a GUI with buttons to click through pick-rounds and to toggle between before/after. At Kairos Logic, a proprietary GUI is used internally (mainly for debugging), but below one project is described which aims at extending the GUI as a service for customers.

The core idea is that the customer should participate in steps involved in digitizing the warehouse and optimizing PRPs, OBPs and SLAPs. One identified problem is that customers often doubt the results of optimization. An interactive GUI provides the means for customers to learn more and to participate in both the digitization and optimization of their warehouse.

Concerning digitization (Section 3.2.1), a GUI can provide the customer with the possibility to add obstacles, unidirectional traffic zones and pick locations manually using drag and drop or a coordinate textbox. After a "submit" button is pressed, the pick-locations are connected, and the shortest paths and distances are generated using the Floyd-Warshall graph algorithm (or similar). After the digitization files have been generated, the GUI shows a picture of all the generated edges.

Concerning optimization, the GUI can help convince the warehouse or WMS customer that their existing operation can be improved. Without the GUI, the customer only has access to an API and documentation describing required and optional fields in an optimization request. After they send the request, they obtain the optimized response, which includes information on the sequence with which products need to be picked, how much distance saving was achieved and how much CPU-time was spent. It is understandable that a customer may not trust some of this information. To help convince them, they can get a visualization of the request and response inside a GUI.

This type of customer-accessible GUI should not be enforced, but rather provided as an optional assistance tool. The customer needs to run some form of integration script which builds the optimization request from information in the WMS, translates this information into a request according to the SaaS API, and sends it to the service URL. After the response has been received, its data needs to be integrated back into the WMS. Forcing the customer to carry out this WMS – SaaS integration in a GUI takes away flexibility on their end.

In summary, a GUI provides opportunities for improved customer relations and satisfaction, but it also strengthens the argument that the developed optimization technology needs to be fast, simple, and flexible (Section 1.7.2). Assuming the business is customer-driven, and that customers have different ideas of how they want optimization to be visualized, the GUI development rate can be expected to be high. The customer only sees the results of

optimization once the optimizer has finished its job, so it is clearly an advantage if it is fast. Apart from time needed for development, one argument against the GUI project is that it risks giving competitors information with which to reverse-engineer features of the SaaS. For a small-scale SaaS, strengthened customer relations arguably outweigh this risk.

4.5 Directed and mixed graphs

The optimization methods described in Section 3.2 assume that the distance matrix is symmetrical, i.e., vehicles can turn on the spot and the distance between two locations is equal. Real warehouses, however, often include areas where vehicles are permitted to travel in only one direction. When a graph of the warehouse is built with vertices and edges, it is *directed* if the edges only permit travel in one direction, and *mixed* if some edges permit travel in one direction and some permit travel in both directions. A brief description of the process with which directed and mixed graphs are handled is provided below.

The digitization process is mainly restricted by the requirements of the Concorde TSP solver, since the proposed optimization methods rely on it to a significant extent. Concorde only works with symmetric distance matrices. For directed and mixed graphs, the distance matrix is instead asymmetric, so a required pre-processing step is to translate the asymmetric matrix to a symmetric one. For details on how this can be done, see Hahsler & Kurt [122]. Briefly, the procedure includes the extension of the asymmetric matrix with dummy vertices and edges with very large or small distances. The dummy vertices are added to the TSP before optimization, and after optimization. The procedure is like the one used for symmetric distances and multi-depot TSPs, but instead of adding one dummy node to the matrix, it adds one dummy node for each location in the TSP.

Directed and mixed graphs are not deemed suitable for standardized PRP, OBP and SLAP optimization. They would be interesting if they significantly affect the performance of TSP optimization (as against bi-directional graphs). For TSP optimization using Simulated Annealing or OR-tools, no difference in performance is observed. For Concorde, some decrease in performance is observed, but it is not deemed significant enough for a closer study. Concorde uses various geometric heuristics, and the usage of dummy vertices and edges with very large distances slows it down in some cases (in the end it always seems to find an optimal solution). Directed or mixed graphs also counteract standardization as they add some complexity in the description of problem instances. The published TSPLIB instances do not include weighted edges or

a distance matrix, as these can instead be produced based on the provided product and obstacle coordinates. Including edges and their directions would add to the size of the instances. One alternative way is to include the directed areas as lists of coordinates with corresponding lists of permitted directions, but this adds complexity. Before further work on directed and mixed graphs is warranted, researchers need to find agreement on what the format for layout-agnostic PRP, OBP and SLAP instances should be. The proposed format of the generated instances is a suggestion, which may be discarded at a future date in favor of a different format.

4.6 TSP optimization using Google maps API

The only non-warehouse POC involves computing paths for washing deliveries using the Google Maps Distance API [123]. This API offers both distance calculations and visualizations of planned outdoor routes. The POC is aimed at seeing whether a light-weight service can be provided based on $200 worth of free monthly requests that the API offers. One limitation of the API is that Google's common TSP optimization service, "optimizeWaypoints", has a maximum of 8 locations (it also costs $0.01 per request). Alternatively, a possibility is to precompute a distance matrix and then optimize the TSP using OR-tools. Google provides a distance matrix API, but it is quite expensive: The API charges per element in the distance matrix and 1000 elements costs $5. This equates to 40000 elements costing $200. In other words, the 200$ quota only suffices to compute a distance matrix with 200 locations. This is not enough to cover all the locations that the washing delivery service may visit in a month. A conclusion drawn is that Google Maps Distance API is most suitable for companies whose delivery vehicles do not make more than 8 stops. This POC also shows that one of the main hurdles when developing routing services outdoors is the construction or purchase of a distance matrix.

5. Conclusion

In this dissertation, we studied how order-picking optimization can be designed and provided as Software as a Service (SaaS) for Warehouse Management Systems (WMS). We investigated three optimization problems related to order-picking:

1. The Picker Routing Problem (PRP), where we optimize the shortest path that a vehicle travels to pick a set of products.
2. The Order Batching Problem (OBP), where we optimize how orders are distributed among a fleet of vehicles, as well as the corresponding PRP for each vehicle.
3. The Storage Location Assignment Problem (SLAP), where we assign or reassign locations for products. After a candidate assignment or reassignment has been found, solution cost is obtained by optimizing PRPs or OBPs.

In related work, there are many proposals for how to select and engineer features for the PRP, OBP and SLAP. Example features include warehouse layouts, locations, dynamicity, stochasticity, capacity and travel constraints and cost functions. We make proposals for how features can be selected and engineered to promote standardization. We use these features to build and publicly share benchmark datasets. We also discuss warehouse digitization and how datastructures can be pre-stored in memory to achieve reduced CPU-times for subsequent PRP, OBP and SLAP optimization.

Concerning optimization, we propose heuristic and meta-heuristic algorithms. For PRP optimization, we primarily use the Concorde TSP solver. Common PRPs rarely exceed a few dozen locations and Concorde is capable of finding solutions to those in fractions of a second. Sub-optimal PRP optimization, in the form of Google OR-tools TSP optimization suite is also effective for common PRPs, as it can find close-to optimal solutions faster than Concorde.

For OBP optimization, we mainly rely on constructive Seed heuristics and Sequential Minimal Distance (SMD) heuristics. We also use Markov Chain

Monte Carlo (MCMC) in the form of a Metropolis algorithm. Our OBP optimizers construct candidate batches and evaluates them by optimizing their corresponding PRPs optimally or sub-optimally. Experiment results show that the Seed and SMD heuristics outperform the Metropolis algorithm. For smaller OBP instances, we found that close-to-optimal results can be achieved within a few seconds. For larger instances, we found that solution improvement quickly slows down. For example, costs obtained after 5-10 seconds are only 4-7% higher than costs obtained after 5 minutes. Warehouse managers often prefer OBP solutions to be obtainable quickly, at a small increase in solution cost.

For periodical SLAP optimization, we found that significant improvements are achievable using MCMC, but that many questions remain with regard to standardization and integration challenges. Apart from questions on what PRP and/or OBP features to include in SLAP optimization models, we have additional features that also need consideration. One such feature is the cost for carrying out a swap between products that are already in the warehouse. Since this type of *reassignment* is optional and not a requirement, a reassignment penalty term needs to be included in the cost function.

Due to the many possible feature combinations in PRPs, OBPs, and SLAPs, as well as varying requirements from WMS customers, integrating an optimization SaaS with them is often challenging. However, customers are generally willing to engage in discussions about order-picking optimization and participate in Proof of Concepts (PoC) as part of consultancy efforts.

The lack of standardization across different systems and processes remains a key challenge. The leveraging of effective optimization methods (including methods utilizing Machine Learning) suffers without standardized data formats and operational protocols. This challenge is emblematic of the interdisciplinary nature of the problem, where computer and warehouse science must converge to deliver solutions that are not only efficient but adaptable to diverse operational environments. For a customer-driven SaaS, the human aspect is also important. Customer requirements are not always practical or aligned with operational realities, and balancing optimization with human factors is critical. Excessive optimization, for instance, can increase safety risks by pushing systems or workers beyond sustainable operational limits. Striking a balance between efficiency, safety, and adaptability is essential, underscoring the need for both technical innovation and pragmatic collaboration.

6. References

[1] J. Bartholdi and S. Hackman, *Warehouse and distribution science Release 0.98*. Supply Chain and Logistics Institute, 2019.

[2] S. Boyd and L. Vandenberghe, *Convex optimization*. Cambridge university press, 2004.

[3] R. de Koster, T. Le-Duc, and K. J. Roodbergen, 'Design and control of warehouse order picking: A literature review', *European Journal of Operational Research*, vol. 182, no. 2, pp. 481–501, 2007.

[4] 'REDCD (creative commons)'. [Online]. Available: https://commons.wikimedia.org/wiki/File:Regional_European_Distribution_Center_in_Duisburg.jpg

[5] S. Henn, S. Koch, and G. Wäscher, 'Order batching in order picking warehouses: a survey of solution approaches', in *Warehousing in the global supply chain*, Springer, 2012, pp. 105–137.

[6] K. Azadeh, R. De Koster, and D. Roy, 'Robotized and Automated Warehouse Systems: Review and Recent Developments', *Transportation Science*, vol. 53, 2019.

[7] 'PACCAR Parts - Lancaster PDC_2 (crative commons)'. 2013. [Online]. Available: https://www.flickr.com/photos/truckpr/9688738292

[8] A. Kamali, 'Smart warehouse vs. traditional warehouse', *CiiT International Journal of Automation and Autonomous System*, vol. 11, no. 1, pp. 9–16, 2019.

[9] J. Kembro and A. Norrman, 'The transformation from manual to smart warehousing: an exploratory study with Swedish retailers', *The International Journal of Logistics Management*, vol. 33, no. 5, pp. 107–135, 2022.

[10] M. van Geest, B. Tekinerdogan, and C. Catal, 'Design of a reference architecture for developing smart warehouses in industry 4.0', *Computers in Industry*, vol. 124, p. 103343, 2020.

[11] M. D. M. Francielly Hedler Staudt Gülgün Alpan and C. M. T. Rodriguez, 'Warehouse performance measurement: a literature review',

International Journal of Production Research, vol. 53, no. 18, pp. 5524–5544, 2015, doi: 10.1080/00207543.2015.1030466.
[12] M. Masae, C. H. Glock, and E. H. Grosse, 'Order picker routing in warehouses: A systematic literature review', *International Journal of Production Economics*, vol. 224, p. 107564, 2020.
[13] H. Ratliff and A. Rosenthal, 'Order-Picking in a Rectangular Warehouse: A Solvable Case of the Traveling Salesman Problem', *Operations Research*, vol. 31, pp. 507–521, 1983.
[14] K. J. Roodbergen and R. Koster, 'Routing methods for warehouses with multiple cross aisles', *International Journal of Production Research*, vol. 39, no. 9, pp. 1865–1883, 2001.
[15] O. Briant, H. Cambazard, D. Cattaruzza, N. Catusse, A.-L. Ladier, and M. Ogier, 'An efficient and general approach for the joint order batching and picker routing problem', *European Journal of Operational Research*, vol. 285, no. 2, pp. 497–512, 2020.
[16] O. Kulak, Y. Sahin, and M. E. Taner, 'Joint order batching and picker routing in single and multiple-cross-aisle warehouses using cluster-based tabu search algorithms', *Flexible Services and Manufacturing Journal*, vol. 24, no. 1, pp. 52–80, 2012.
[17] D. Jungnickel, 'Basic graph theory', *Graphs, Networks and Algorithms*, pp. 1–33, 2013.
[18] M. Garfinkel, 'Minimizing multi-zone orders in the correlated storage assingment problem', PhD Thesis, School of Industrial and Systems Engineering, Georgia Institute of Technology, 2005.
[19] 'Picking Trolley (crative commons)'. 2019. [Online]. Available: https://commons.wikimedia.org/wiki/File:Online_Groceries_Picking_Trolley.jpg
[20] 'TMHE's New Order Picker – BT Optio OSE250 (creative commons)'. 2011. [Online]. Available: https://www.flickr.com/photos/toyotamheurope/6284072035
[21] X. Xiang, C. Liu, and L. Miao, 'Storage assignment and order batching problem in Kiva mobile fulfilment system', *Engineering Optimization*, vol. 50, no. 11, pp. 1941–1962, 2018, doi: 10.1080/0305215X.2017.1419346.
[22] C. A. Valle, J. E. Beasley, and A. S. da Cunha, 'Optimally solving the joint order batching and picker routing problem', *European Journal of Operational Research*, vol. 262, no. 3, pp. 817–834, 2017.
[23] E. Charris, J. Rojas-Reyes, and J. Montoya-Torres, 'The storage location assignment problem: A literature review', *International Journal of Industrial Engineering Computations*, vol. 10, 2018.

[24] P. Kübler, C. H. Glock, and T. Bauernhansl, ‘A new iterative method for solving the joint dynamic storage location assignment, order batching and picker routing problem in manual picker-to-parts warehouses’, *Computers & Industrial Engineering*, vol. 147, p. 106645, 2020.
[25] J. Oxenstierna, L. J. van Rensburg, P. J. Stuckey, and V. Krueger, ‘Optimization of the Storage Location Assignment Problem Using Nested Annealing’, in *International Conference on Operations Research and Enterprise Systems*, in Communications in Computer and Information Science book series, vol. 1985. Springer, 2022, pp. 220–244. [Online]. Available: https://www.springer.com/series/7899
[26] T. Münsberg, L. Hvam, S. Lundsteen, M. Støjfer-Hønberg, M. Csik, and L. Tsintzou, ‘Four Initiatives to Standardize Warehouses to Increase Digitalization and Automation’, in *2022 IEEE International Conference on Industrial Engineering and Engineering Management (IEEM)*, IEEE, 2022, pp. 1164–1168.
[27] M. Dotoli, N. Epicoco, M. Falagario, N. Costantino, and B. Turchiano, ‘An integrated approach for warehouse analysis and optimization: A case study’, *Computers in Industry*, vol. 70, pp. 56–69, 2015.
[28] C. E. Shalley and L. L. Gilson, ‘Creativity and the management of technology: Balancing creativity and standardization’, *Production and Operations Management*, vol. 26, no. 4, pp. 605–616, 2017.
[29] C. A. Voss, P. Åhlström, and K. Blackmon, ‘Benchmarking and operational performance: some empirical results’, *International Journal of Operations & Production Management*, vol. 17, no. 10, pp. 1046–1058, 1997.
[30] J. Mańdziuk and M. Świechowski, ‘UCT in Capacitated Vehicle Routing Problem with traffic jams’, *Information Sciences*, vol. 406–407, pp. 42–56, 2017.
[31] M. Okulewicz and J. Mańdziuk, ‘The impact of particular components of the PSO-based algorithm solving the Dynamic Vehicle Routing Problem’, *Applied Soft Computing*, vol. 58, pp. 586–604, 2017.
[32] V. Pillac, M. Gendreau, C. Guéret, and A. L. Medaglia, ‘A review of dynamic vehicle routing problems’, *European Journal of Operational Research*, vol. 225, no. 1, pp. 1–11, 2013.
[33] H. Psaraftis, M. Wen, and C. Kontovas, ‘Dynamic Vehicle Routing Problems: Three Decades and Counting’, *Networks*, vol. 67, 2015.
[34] P. M. Morse, G. E. Kimball, and S. I. Gass, *Methods of operations research*. Courier Corporation, 2003.

[35] A. Thesen, *Computer methods in operations research*. Academic Press, 2014.
[36] A. H. Eden, ‘Three paradigms of computer science’, *Minds and machines*, vol. 17, pp. 135–167, 2007.
[37] B. D. Williams and T. Tokar, ‘A review of inventory management research in major logistics journals: Themes and future directions’, *The International Journal of Logistics Management*, vol. 19, no. 2, pp. 212–232, 2008.
[38] S. T. W. Mara, R. Kuo, and A. M. S. Asih, ‘Location-routing problem: a classification of recent research’, *International Transactions in Operational Research*, vol. 28, no. 6, pp. 2941–2983, 2021.
[39] M. Mansouri, F. Lagriffoul, and F. Pecora, ‘Multi Vehicle Routing with Nonholonomic Constraints and Dense Dynamic Obstacles’, 2017.
[40] G. Nagy and S. Salhi, ‘Location-routing: Issues, models and methods’, *European Journal of Operational Research*, vol. 177, no. 2, pp. 649–672, 2007, doi: https://doi.org/10.1016/j.ejor.2006.04.004.
[41] N. Gademann and V. de S. Velde, ‘Order batching to minimize total travel time in a parallel-aisle warehouse’, *IIE Transactions*, vol. 37, no. 1, pp. 63–75, 2005.
[42] Y. A. Bozer and J. W. Kile, ‘Order batching in walk-and-pick order picking systems’, *International Journal of Production Research*, vol. 46, no. 7, pp. 1887–1909, 2008.
[43] M. Bortolini, M. Faccio, E. Ferrari, M. Gamberi, and F. Pilati, ‘Design of diagonal cross-aisle warehouses with class-based storage assignment strategy’, *The International Journal of Advanced Manufacturing Technology*, vol. 100, no. 9, pp. 2521–2536, Feb. 2019.
[44] E. Cogo, E. Žunić, A. Beširević, S. Delalić, and K. Hodžić, ‘Position based visualization of real world warehouse data in a smart warehouse management system’, in *2020 19th International Symposium INFOTEH-JAHORINA (INFOTEH)*, IEEE, 2020, pp. 1–6.
[45] E. Zunic, A. Besirevic, R. Skrobo, H. Hasic, K. Hodzic, and A. Djedovic, ‘Design of optimization system for warehouse order picking in real environment’, in *2017 XXVI International Conference on Information, Communication and Automation Technologies (ICAT)*, IEEE, 2017, pp. 1–6.
[46] M. Yu and R. B. M. de Koster, ‘The impact of order batching and picking area zoning on order picking system performance’, *European Journal of Operational Research*, vol. 198, no. 2, pp. 480–490, 2009.
[47] M. Kofler, A. Beham, S. Wagner, and M. Affenzeller, ‘Affinity Based Slotting in Warehouses with Dynamic Order Patterns’, no. Advanced

Methods and Applications in Computational Intelligence, pp. 123–143, 2014.
[48] M.-C. Chen and H.-P. Wu, ‘An association-based clustering approach to order batching considering customer demand patterns’, *Omega*, vol. 33, no. 4, pp. 333–343, 2005.
[49] P. Jahani, ‘Dynamic warehouse optimization using predictive analytics.’, *Electronic Theses and Dissertations. Paper 2582.*, 2016, doi: https://doi.org/10.18297/etd/2582.
[50] N. C. Truong, T. G. Dang, and D. A. Nguyen, ‘Building management algorithms in automated warehouse using continuous cluster analysis method’, in *AETA 2017-Recent Advances in Electrical Engineering and Related Sciences: Theory and Application*, Springer, 2018, pp. 1068–1077.
[51] M. B. M. D. Koster, E. S. V. der Poort, and M. Wolters, ‘Efficient orderbatching methods in warehouses’, *International Journal of Production Research*, vol. 37, no. 7, pp. 1479–1504, 1999.
[52] S. Henn, ‘Algorithms for on-line order batching in an order picking warehouse’, *Computers & Operations Research*, vol. 39, no. 11, pp. 2549–2563, 2012.
[53] P. Kilby, P. Prosser, and P. Shaw, ‘Dynamic VRPs: A study of scenarios’, *University of Strathclyde Technical Report*, vol. 1, no. 11, 1998.
[54] V. Beiranvand, W. Hare, and Y. Lucet, ‘Best practices for comparing optimization algorithms’, *Optimization and Engineering*, vol. 18, pp. 815–848, 2017.
[55] I. Žulj, C. H. Glock, E. H. Grosse, and M. Schneider, ‘Picker routing and storage-assignment strategies for precedence-constrained order picking’, *Computers & Industrial Engineering*, vol. 123, pp. 338–347, 2018, doi: https://doi.org/10.1016/j.cie.2018.06.015.
[56] I. G. Lee, S. H. Chung, and S. W. Yoon, ‘Two-stage storage assignment to minimize travel time and congestion for warehouse order picking operations’, *Computers & Industrial Engineering*, vol. 139, p. 106129, 2020, doi: https://doi.org/10.1016/j.cie.2019.106129.
[57] I. Nieuwenhuyse, R. De Koster, and J. Colpaert, ‘Order batching in multi-server pick-and-sort warehouses’, *Katholieke Universiteit Leuven, Open Access publications from Katholieke Universiteit Leuven*, 2007.
[58] G. P. Sharp and D. R. Gibson, ‘Order batching procedures’, *European Journal of Operational Research*, no. 58, 1992.

[59] K. Braekers, K. Ramaekers, and I. V. Nieuwenhuyse, ‘The vehicle routing problem: State of the art classification and review’, *Computers & Industrial Engineering*, vol. 99, pp. 300–313, 2016.
[60] Y. Li, R. Zhang, and D. Jiang, ‘Order-Picking Efficiency in E-Commerce Warehouses: A Literature Review’, *Journal of Theoretical and Applied Electronic Commerce Research*, vol. 17, no. 4, pp. 1812–1830, 2022.
[61] E. G. Pardo, S. Gil-Borrás, A. Alonso-Ayuso, and A. Duarte, ‘Order batching problems: Taxonomy and literature review’, *European Journal of Operational Research*, vol. 313, no. 1, pp. 1–24, 2024.
[62] R. Mantel, P. Schuur, and S. Heragu, ‘Order oriented slotting: A new assignment strategy for warehouses’, *European Journal of Industrial Engineering*, vol. 1, pp. 301–316, 2007.
[63] B. S. Kim and J. S. Smith, ‘Slotting methodology using correlated improvement for a zone-based carton picking distribution system’, *Computers & Industrial Engineering*, vol. 62, no. 1, pp. 286–295, 2012.
[64] N. Boysen and K. Stephan, ‘The deterministic product location problem under a pick-by-order policy’, *Discrete Applied Mathematics*, vol. 161, no. 18, pp. 2862–2875, 2013.
[65] J. Gu, M. Goetschalckx, and L. F. McGinnis, ‘Research on warehouse operation: A comprehensive review’, *European journal of operational research*, vol. 177, no. 1, pp. 1–21, 2007.
[66] D. Ming-Huang Chiang, C.-P. Lin, and M.-C. Chen, ‘Data mining based storage assignment heuristics for travel distance reduction’, *Expert Systems*, vol. 31, no. 1, pp. 81–90, 2014.
[67] J.-F. Cordeau, G. Laporte, M. Savelsbergh, and D. Vigo, ‘Vehicle Routing’, in *Transportation, handbooks in operations research and management science*, vol. 14, 2007, pp. 195–224.
[68] T. Hintsch and S. Irnich, ‘Exact solution of the soft-clustered vehicle-routing problem’, *European Journal of Operational Research*, vol. 280, no. 1, pp. 164–178, 2020.
[69] M. Bué, D. Cattaruzza, M. Ogier, and F. Semet, ‘A Two-Phase Approach for an Integrated Order Batching and Picker Routing Problem’, 2019, pp. 3–18.
[70] G. Reinelt, ‘TSPLIB - A Traveling Salesman Problem Library’, *INFORMS J. Comput.*, vol. 3, pp. 376–384, 1991.
[71] W. Cook, ‘TSP test data’. 2009. [Online]. Available: https://www.math.uwaterloo.ca/tsp/data/index.html

[72] M. H. Bhoir and M. R. P. Principal, ‘Cloud computing for supply chain management’, *International Journal of Innovations in Engineering Research and Technology*, vol. 1, no. 2, pp. 1–9, 2014.
[73] G. A. Raj, M. K. Sampath, and V. Venkatesh, ‘Opportunities for Cloud Based Software as a Service (SaaS) Warehouse Management System an Indian Industry Insight’, *SAMVAD*, vol. 6, no. 2, pp. 43–60, 2013.
[74] N. Andiyappillai, ‘Factors Influencing the Successful Implementation of the Warehouse Management System (WMS)’, *International Journal of Computer Applications*, vol. 177, pp. 21–25, 2020.
[75] V. N. H. Nguyen, ‘SaaS, IaaS, and PaaS: Cloud-computing in Supply Chain Management. Case study: Food Service Ltd.’, 2021.
[76] L. Novais, J. M. Maqueira, and Á. Ortiz-Bas, ‘A systematic literature review of cloud computing use in supply chain integration’, *Computers & Industrial Engineering*, vol. 129, pp. 296–314, 2019, doi: https://doi.org/10.1016/j.cie.2019.01.056.
[77] C. Esposito, A. Castiglione, and K.-K. R. Choo, ‘Challenges in Delivering Software in the Cloud as Microservices’, *IEEE Cloud Computing*, vol. 3, no. 5, pp. 10–14, 2016, doi: 10.1109/MCC.2016.105.
[78] J. Innerbichler, S. Gonul, V. Damjanovic-Behrendt, B. Mandler, and F. Strohmeier, ‘NIMBLE collaborative platform: Microservice architectural approach to federated IoT’, in *2017 Global Internet of Things Summit (GIoTS)*, IEEE, 2017, pp. 1–6.
[79] I. Gunawan, W. Witanti, and F. Renaldi, ‘Integration of Supply Management System in Auto Parts Company Using Web Services’, in *Journal of Physics: Conference Series*, IOP Publishing, 2021, p. 012022.
[80] D. Rani and R. K. Ranjan, ‘A comparative study of SaaS, PaaS and IaaS in cloud computing’, *International Journal of Advanced Research in Computer Science and Software Engineering*, vol. 4, no. 6, 2014.
[81] L. F. Albuquerque Jr, F. S. Ferraz, R. Oliveira, and S. Galdino, ‘Function-as-a-service x platform-as-a-service: Towards a comparative study on FaaS and PaaS’, in *ICSEA*, 2017, pp. 206–212.
[82] C. E. Miller, A. W. Tucker, and R. A. Zemlin, ‘Integer programming formulation of traveling salesman problems’, *Journal of the ACM (JACM)*, vol. 7, no. 4, pp. 326–329, 1960.
[83] A. Scholz, S. Henn, M. Stuhlmann, and G. Wäscher, ‘A new mathematical programming formulation for the single-picker routing problem’, *European Journal of Operational Research*, vol. 253, no. 1, pp. 68–84, 2016.

[84] K.-W. Pang and H.-L. Chan, ‘Data mining-based algorithm for storage location assignment in a randomised warehouse’, *International Journal of Production Research*, vol. 55, no. 14, pp. 4035–4052, 2017.
[85] R. de Koster and E. V. D. Poort, ‘Routing orderpickers in a warehouse: a comparison between optimal and heuristic solutions’, *IIE transactions*, vol. 30, no. 5, pp. 469–480, 1998.
[86] W. Cook, ‘Concorde TSP Solver’. 2020. [Online]. Available: http://www.math.uwaterloo.ca/tsp/concorde/index.html
[87] J. Oxenstierna, J. Malec, and V. Krueger, ‘Layout-Agnostic Order-Batching Optimization’, in *International Conference on Computational Logistics*, Springer, 2021, pp. 115–129.
[88] D. Applegate, W. Cook, S. Dash, and A. Rohe, ‘Solution of a Min-Max Vehicle Routing Problem’, *INFORMS Journal on Computing*, vol. 14, pp. 132–143, 2002.
[89] A. Fumi, L. Scarabotti, and M. Schiraldi, ‘The Effect of Slot-Code Optimization in Warehouse Order Picking’, *International Journal of Business and Management*, vol. 5, 2013.
[90] S. Henn and G. Wäscher, ‘Tabu search heuristics for the order batching problem in manual order picking systems’, *European Journal of Operational Research*, vol. 222, no. 3, pp. 484–494, 2012.
[91] Y.-C. Ho, T.-S. Su, and Z.-B. Shi, ‘Order-batching methods for an order-picking warehouse with two cross aisles’, *Computers & Industrial Engineering*, vol. 55, no. 2, pp. 321–347, 2008.
[92] J. Li, R. Huang, and J. B. Dai, ‘Joint optimisation of order batching and picker routing in the online retailer’s warehouse in China’, *International Journal of Production Research*, vol. 55, no. 2, 2017.
[93] C. Cergibozan and A. Tasan, ‘Genetic algorithm based approaches to solve the order batching problem and a case study in a distribution center’, *Journal of Intelligent Manufacturing*, pp. 1–13, 2020.
[94] B. Aerts, T. Cornelissens, and K. Sörensen, ‘The joint order batching and picker routing problem: Modelled and solved as a clustered vehicle routing problem’, *Computers & Operations Research*, vol. 129, p. 105168, 2021, doi: https://doi.org/10.1016/j.cor.2020.105168.
[95] C. Kallina and J. Lynn, ‘Application of the Cube-per-Order Index Rule for Stock Location in a Distribution Warehouse’, *Interfaces*, vol. 7, no. 1, pp. 37–46, 1976.
[96] R.-Q. Zhang, M. Wang, and X. Pan, ‘New model of the storage location assignment problem considering demand correlation pattern’, *Computers & Industrial Engineering*, vol. 129, pp. 210–219, 2019, doi: https://doi.org/10.1016/j.cie.2019.01.027.

[97] L. Yingde and J. S. Smith, ‘Dynamic slotting optimization based on skus correlations in a zone-based wave-picking system’, 2012.
[98] L. Begnardi, H. Baier, W. van Jaarsveld, and Y. Zhang, ‘Deep Reinforcement Learning for Two-sided Online Bipartite Matching in Collaborative Order Picking’, in *Asian Conference on Machine Learning*, PMLR, 2024, pp. 121–136.
[99] C. Seward, ‘Optimizing Warehouse Operations with Machine Learning on GPUs’, Nvidia Developer. [Online]. Available: https://developer.nvidia.com/blog/optimizing-warehouse-operations-machine-learning-gpus/
[100] J. Oxenstierna, ‘Warehouse vehicle routing using deep reinforcement learning’. Uppsala University M.Sc. Thesis, 2019.
[101] G. Dunn, H. Charkhgard, A. Eshragh, S. Mahmoudinazlou, and E. Stojanovski, ‘Deep Reinforcement Learning for Picker Routing Problem in Warehousing’, *arXiv preprint arXiv:2402.03525*, 2024.
[102] D. Silver *et al.*, ‘Mastering the game of go without human knowledge’, *nature*, vol. 550, no. 7676, pp. 354–359, 2017.
[103] O. Vinyals *et al.*, ‘Grandmaster level in StarCraft II using multi-agent reinforcement learning’, *Nature*, vol. 575, no. 7782, pp. 350–354, 2019.
[104] C. Berner *et al.*, ‘Dota 2 with large scale deep reinforcement learning’, *arXiv preprint arXiv:1912.06680*, 2019.
[105] D. Google, *AlphaStar public repository*. (2017). [Online]. Available: https://github.com/google-deepmind/alphastar
[106] O. Vinyals, M. Fortunato, and N. Jaitly, ‘Pointer networks’, *Advances in neural information processing systems*, vol. 28, 2015.
[107] J. Oxenstierna, J. Malec, and V. Krueger, ‘Storage Assignment Using Nested Metropolis Sampling and Approximations of Order Batching Travel Costs’, *SN Computer Science*, vol. 5, no. 5, p. 477, Apr. 2024, doi: 10.1007/s42979-024-02711-w.
[108] E. Spiliotis, S. Makridakis, A.-A. Semenoglou, and V. Assimakopoulos, ‘Comparison of statistical and machine learning methods for daily SKU demand forecasting’, *Operational Research*, vol. 22, no. 3, pp. 3037–3061, 2022.
[109] M. Seyedan and F. Mafakheri, ‘Predictive big data analytics for supply chain demand forecasting: methods, applications, and research opportunities’, *Journal of Big Data*, vol. 7, no. 1, p. 53, 2020.
[110] M. Abolghasemi, E. Beh, G. Tarr, and R. Gerlach, ‘Demand forecasting in supply chain: The impact of demand volatility in the presence of promotion’, *Computers & Industrial Engineering*, vol. 142, p. 106380, 2020, doi: https://doi.org/10.1016/j.cie.2020.106380.

[111] C. S. Bojer and J. P. Meldgaard, ‘Kaggle forecasting competitions: An overlooked learning opportunity’, *International Journal of Forecasting*, vol. 37, no. 2, pp. 587–603, 2021, doi: https://doi.org/10.1016/j.ijforecast.2020.07.007.
[112] Y. Freund and R. E. Schapire, ‘Experiments with a New Boosting Algorithm’. 1996.
[113] K. Hodžić, H. Hasić, E. Cogo, and Ž. Jurić, ‘Warehouse demand forecasting based on long short-term memory neural networks’, in *2019 XXVII International Conference on Information, Communication and Automation Technologies (ICAT)*, IEEE, 2019, pp. 1–6.
[114] J. Oxenstierna, L. J. van Rensburg, J. Malec, and V. Krueger, ‘Formulation of a Layout-Agnostic Order Batching Problem’, in *Optimization and Learning*, B. Dorronsoro, L. Amodeo, M. Pavone, and P. Ruiz, Eds., Cham: Springer International Publishing, 2021, pp. 216–226.
[115] C. Xie and J. Duthie, ‘An excess-demand dynamic traffic assignment approach for inferring origin-destination trip matrices’, *Networks and Spatial Economics*, vol. 15, pp. 947–979, 2015.
[116] J. Won and S. Olafsson *, ‘Joint order batching and order picking in warehouse operations’, *International Journal of Production Research*, vol. 43, no. 7, pp. 1427–1442, 2005, doi: 10.1080/00207540410001733896.
[117] L. J. van Janse van Rensburg, ‘Artificial intelligence for warehouse picking optimization - an NP-hard problem’, Master’s Thesis, Uppsala University, 2019.
[118] S. Kruk, *Practical Python AI Projects: Mathematical Models of Optimization Problems with Google OR-Tools*. Apress, 2018.
[119] R. Noshy, *Traveling Salesman Optimization*. [Online]. Available: https://github.com/rameziophobia/Travelling_Salesman_Optimization
[120] J. Oxenstierna, J. Malec, and V. Krueger, ‘Efficient order batching optimization using seed heuristics and the metropolis algorithm’, *SN Computer Science*, vol. 4, no. 2, p. 107, 2022.
[121] B. Menéndez, E. G. Pardo, A. Alonso-Ayuso, E. Molina, and A. Duarte, ‘Variable neighborhood search strategies for the order batching problem’, *Computers & Operations Research*, vol. 78, pp. 500–512, 2017.
[122] M. Hahsler and H. Kurt, ‘TSP – Infrastructure for the Traveling Salesperson Problem’, *Journal of Statistical Software*, vol. 2, pp. 1–21, 2007.

[123] Google Maps Platform, *Google Maps Distance API*. [Online]. Available: https://developers.google.com/maps/documentation/distance-matrix

7. Papers

Contribution Statement

For each paper, Johan Oxenstierna provided an initial version, which was then revised to the final form in collaboration with co-authors. Johan Oxenstierna carried out at least 90% of the implementation work in all papers.

Formulation of a Layout-Agnostic Order Batching Problem

Johan Oxenstierna, Louis Janse van Rensburg, Jacek Malec and Volker Krueger

Abstract

To date, research on warehouse order-batching has been limited by reliance on rigid assumptions regarding rack layouts. Although efficient optimization algorithms have been provided for conventional warehouse layouts with Manhattan style blocks of racks, they are limited in that they fail to generalize to unconventional layouts. This paper builds on a generalized procedure for digitization of warehouses where racks and other obstacles are defined using two-dimensional polygons. We extend on this digitization procedure to introduce a layout-agnostic minisum formulation for the Order Batching Problem (OBP), together with a sub-problem for the OBP for a single vehicle, the *single batch* OBP. An algorithm which optimizes the *single batch* OBP iteratively until an approximate solution to the OBP can be obtained, is discussed. The formulations will serve as the fundament for further work on layout-agnostic OBP optimization and generation of benchmark datasets. Experimental results for the digitization process involving various settings are presented.

1. Introduction

Order-picking is “the process of retrieving products from storage areas in response to a specific customer request” where “customer request” denotes a shipment order consisting of one or several products [1]. Order-picking is

accountable for as much as 55% of all operating expenses in a warehouse and is considered an important process to optimize [2]. *Order-batching* is a common method with which to conduct order-picking. It means that each picker (vehicle) is set to pick a so-called *batch* of one or more orders [3]. As an optimization problem order-batching is known as the *Order Batching Problem* (OBP) [4] or the *Joint Order Batching and Picker Routing Problem* (JOBPRP) [5]. The *Picker Routing Problem* is a sub-problem of the OBP for one vehicle and is here treated as equivalent to the Traveling Salesman Problem (TSP) [6]. This paper follows the convention that an "OBP" can include TSP optimization without having to include TSP optimization in the name of the problem (such as the JOBPRP) [4]. The *Picker Routing Problem* is henceforth referred to as TSP and the *Order Batching Problem*, which includes TSP optimization, as OBP. In the literature the OBP is usually formulated as a specific form of the more well-known Vehicle Routing Problem (VRP) [7], with two key amendments:

- *Order-integrity*: In the OBP products in one order cannot be picked by more than one vehicle [8] whereas in the VRP this constraint is not used (there is no notion of a warehouse shipment "order" in the VRP) [7].
- *Obstacle-layout:* We can observe two types of obstacle layouts (see Fig 1): In the *conventional* layout, racks are laid out in a Manhattan style blocks. In the *unconventional* layout, racks or other obstacles can be freely placed (see Fig 2. for examples). The *unconventional* layout includes the case when there are no racks or obstacles at all. All previous work on the OBP seems to require explicitly a conventional layout [5], [8]–[10], while the VRP does not have this requirement.

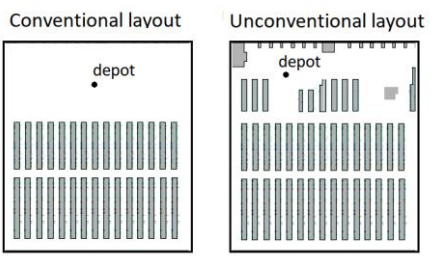

Fig. 1. Example of a *conventional* layout (left) with 30 *racks*, 16 *aisles* and 3 *cross-aisles*. Adding a single or a few irregular racks or other obstacles to the *conventional* layout renders it *unconventional*.

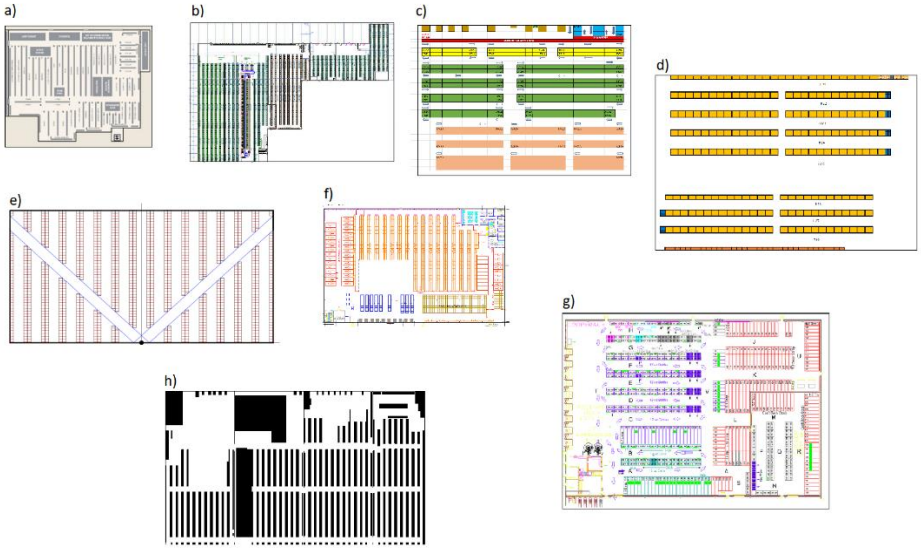

Fig. 2. Eight examples of unconventional warehouse layouts. a) and b) show cases where the layout has been built to fit within a non-rectangular outer wall. e) is the so called "fishbone" layout.

The aim of this paper is to formulate an OBP where orders and order-integrity are preserved, but where the layout is generalized towards any layout with or without polygonal obstacles. This is in line with a future research recommendation by Masae et al. [11]: "there is a strong need for developing […] algorithms for […] non-conventional warehouses". Below are some reasons for why this is important:

- It allows warehouses with unconventional layouts to formulate and optimize OBP's. This includes warehouses divided into zones where each zone has a conventional layout.
- It allows OBP optimization to be used as a tool with which to optimize warehouse layouts beyond conventional layouts.
- Problems in non-warehouse domains, such as agriculture, mining, road and aerial logistics to be explored as OBP's. The OBP is fundamentally similar to *batch processing* [12] where each process

consists of constrained sub-processes (similar to *order-integrity*), and the Key Performance Indicator (KPI) depends on how well the sub-processes operate when they are combined. These types of broadened perspectives on the OBP can only be pursued if it is generalized beyond conventional layouts.

The paper continues with a literature review (Section 2), followed by the OBP formulation (Section 3). The formulation builds on a digitization process which generates the distances and shortest paths between all defined locations for a given warehouse [13]. The feasibility of the digitization process is examined in experiments involving various warehouse configurations (Section 4).

2. Literature Review

The OBP is a specific form of the Vehicle Routing Problem (VRP) [7] and a specific VRP-variant known as the *Steiner*-VRP [14]. A key feature of the *Steiner*-VRP is that multiple visits to same location (representing a vertex in a graph) are allowed [5], [8], [10], [14]. OBP's and VRP's are known to be NP-hard [15], [16]. OBP's have been formulated using integer programming (e.g. [14]) or set-partitioning (e.g. [4]), with a heavy reliance on heuristics for a *conventional* warehouse layout. The conventional layout is modeled such that obstacles (racks) are arranged with parallel "aisles" (between racks) and parallel "cross-aisles" (between sections of racks) [9], [14]. Using such restrictive definitions for aisles and cross-aisles makes it possible to formulate heuristics that reduce the solution space of an OBP. Briant et al. [9], for example, use cutting planes and various relaxation heuristics to formulate an OBP which they then propose optimality bounds for. They use a conventional layout with 8 aisles and 3 cross-aisles, which corresponds to the size of the warehouse shown in Fig. 2 d).

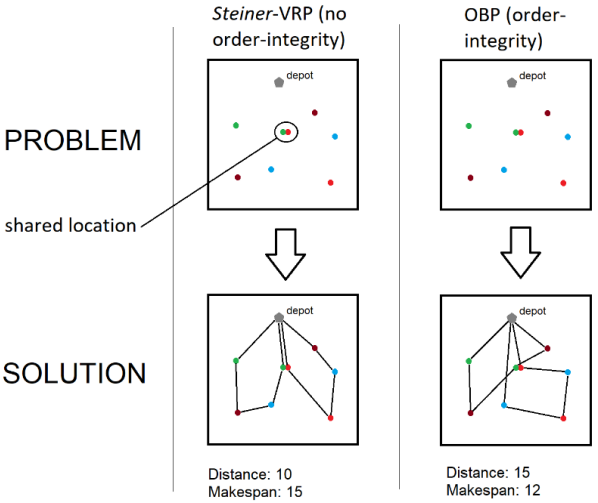

Fig. 3. A Steiner-VRP (left) plotted against the proposed layout-agnostic OBP in a setting without any obstacles. The dots denote products and the colors orders which the products belong to. The outlined green and red products in the middle share the same location. The difference between the Steiner-VRP and the OBP seen here is solely due to the order-integrity constraint. The vehicle distances may be longer in the OBP but the products which they are assigned to carry are more associated (by order color in this example). Order-integrity is used to e.g. reduce a later time-consuming sorting effort or to reduce pick-error i.e. the risk of the wrong product going into the wrong order.

The conventional layout appears in formulations as "number of aisles" [8], "the cross-distance between two consecutive aisles" [4], "number of vertices in the subaisle" [14] or "intra-aisle distance" [17]. They are used as required inputs for OBP optimization. Some authors have called for formulations involving more layouts than the conventional layout [11], [18]–[21]. Without the conventional layout, however, it is a challenging task to effectively constrain an OBP solution space. This can for instance be exemplified in the scenario when there are no obstacles, and each order contains a single product. In that case the OBP is equivalent to a Steiner-VRP, and this problem has no yet proposed optimal solution [14]. Proposed OBP optimization algorithms for the conventional layout include dynamic programming [9], datamining [22], clustering [10] and meta-heuristics such as Tabu Search [23], Ant Colony Optimization [15] and Genetic Algorithms [24]. In the VRP research domain problem formulations are generally not concerned with obstacle layouts [25]. Instead the only requirement in a VRP is usually a cost matrix, providing the travel distance or time between all pairs of locations [7], [26]. In a VRP it is generally assumed that this cost matrix already exists, or that it is produced in

a prior data collection process. In research on the OBP, on the other hand, plenty of attention is usually given to how to produce the cost matrix and how to define shortest paths or TSP's in an environment with obstacles. This can also be seen in some papers on VRP's that include obstacles (e.g. [27] and [28]). Concerning where vehicles begin and end their trips, most OBP papers assume that the origin and destination location is the same (usually this location is named *depot*). If this is not the case, the OBP is denoted *multi-depot* or a *Dial-A-Ride-Problem* (DARP) [21]. An example of this is when vehicles have one location where they drop off their picked orders, and where there are one or several locations where they can start their rides.

3. Problem Formulation

3.1 Preliminaries

The proposed OBP formulation is based on an undirected, symmetric and weighted graph. Without obstacles (racks or other) no graph is needed since distances between all pairs of locations in that case can be assumed to be Euclidean. Also, in the obstacle free case, the shortest path between any two locations can be assumed to be a single edge. With obstacles, however, shortest distances must be calculated based on the shortest paths that circumvent obstacles, and this is achieved here using the Floyd-Warshall graph algorithm [13], [29]. Concerning number of depots the below formulation assumes both an origin and a destination location for vehicles is formulated (but they can share the same coordinates).

First a set of locations is defined as $\mathcal{L} \subset \mathbb{R}^+ \times \mathbb{R}^+$. This set consists of different types of locations: $l_s \in \mathcal{L}$ is the starting (origin) location for all vehicles. $l_d \in \mathcal{L}$ is the destination location for all vehicles. $\mathcal{L}_\mathcal{P} \subset \mathcal{L}$ is the set of product locations. $\mathcal{L}_\mathcal{U} \subset \mathcal{L}$ is a union of sets of obstacles: $\mathcal{L}_\mathcal{U} = \cup_i u_i, i \in \mathbb{N}^+$ where each u_i is a polygonal obstacle with a set of corner locations $u_i = \{l_i^1, l_i^2, \dots, l_i^k\} \subseteq \mathcal{L}_\mathcal{U}, k \in \mathbb{N}^+$. All of the locations can thus be summarized as a union: $\mathcal{L} = \{l_s\} \cup \{l_d\} \cup \mathcal{L}_\mathcal{P} \cup \mathcal{L}_\mathcal{U}$. The products which are to be collected are defined as a set $\mathcal{P} = \{p_1, p_2, \dots, p_n\}, n \in \mathbb{N}^+$. Each product $p \in \mathcal{P}$ has a location $loc^p: \mathcal{P} \to \mathcal{L}_\mathcal{P}$, weight $w^p: \mathcal{P} \to \mathbb{R}^+$ and volume $vol^p: \mathcal{P} \to \mathbb{R}^+$. *The unassigned orders* which are to be batched are defined as a subset of all

possible combinations of products $\mathcal{O} \subset 2^{\mathcal{P}}$. The locations of the products in an order $o \in \mathcal{O}$ are defined as a function $loc^o: \mathcal{O} \to 2^{\mathcal{L}_\mathcal{P}}$. Order weight and volume quantities are defined as $w^o: \mathcal{O} \to \mathbb{R}^+$ and $vol^o: \mathcal{O} \to \mathbb{R}^+$. $w(o) = \sum_{p \in o} w(p), vol(o) = \sum_{p \in o} vol(p)$. *Vehicles* are defined as $\mathcal{M} = \{(w, vol, k, id) | \, w, vol, id \in \mathbb{R}^+, k \in \mathbb{N}^+\}$ where w denotes weight capacity, vol denotes volume capacity, k denotes the maximum number of orders the vehicle can carry and id a unique identifier of a vehicle. The capacities of a single vehicle $m \in \mathcal{M}$ are provided using functions $w^m: \mathcal{M} \to \mathbb{R}^+, vol^m: \mathcal{M} \to \mathbb{R}^+$ and $k^m: \mathcal{M} \to \mathbb{N}^+$.

The digital model of the warehouse is represented as a graph with a set of *vertices* $\mathcal{V} = \{v_1, v_2, \dots, v_n\}, n = |\mathcal{L}|$. $\mathcal{V}$ consists of different types of vertices denoted as follows: $v_s \in \mathcal{V}$ is a starting (origin) vertex for vehicles, $v_d \in \mathcal{V}$ is a destination vertex for vehicles, $\mathcal{V}_{\mathcal{L}_\mathcal{P}} \subset \mathcal{V}$ is a set of product location vertices and $\mathcal{V}_\mathcal{U} \subset \mathcal{V}$ is a set of obstacle corner vertices. The union of all vertices, $\mathcal{V} = \{v_s\} \cup \{v_d\} \cup \mathcal{V}_{\mathcal{L}_\mathcal{P}} \cup \mathcal{V}_\mathcal{U}$, are defined similarly to the locations apart from one important difference: There may be several products in one location and there is one vertex per *product location*, not one vertex per product (this is to limit the size of the graph). To get a set of locations from a corresponding set of vertices the function $loc^\mathcal{V}: \mathcal{V} \to \mathcal{L}$ is used. To get a set of vertices from a set of locations is similarly provided by the function $v^\mathcal{L}: \mathcal{L} \to \mathcal{V}$.

The set of possible batches is defined as $\mathcal{B} \subset 2^\mathcal{O}, b \in \mathcal{B}, b \in 2^\mathcal{O}, b \neq \emptyset$. The locations of the products in the batch can be obtained using function $loc^b: \mathcal{B} \to 2^{\mathcal{L}_\mathcal{P}}$. $loc(b) = \cup_{o \in b} loc(o)$. Similarly, the vertices in the batch are $v^b: \mathcal{B} \to 2^{\mathcal{V}_{\mathcal{L}_\mathcal{P}}}$. $v(b) = \cup_{o \in b} v(loc(o))$. Batch weight and volume quantities are defined as $w^b: \mathcal{B} \to \mathbb{R}^+$ and $vol^b: \mathcal{B} \to \mathbb{R}^+$. The number of orders in a batch is defined as $k^b: \mathcal{B} \to \mathbb{N}^+$ or $|b|$.

The set of edges E is defined such that each edge is an ordered pair $e \in E = \{(i, j), i, j \in \mathcal{V}, \mathrm{i} \neq j\}$ where i is an origin and j a destination vertex. E excludes any edge which passes through the hull of any polygon in $\mathcal{U}$ (for details on how this can be achieved see [13]). Edges between adjacent corners in any polygon $u \in \mathcal{U}$ are not excluded in E. The edges and vertices are then used to construct the *symmetric undirected weighted graph* $G = (\mathcal{V}, E)$.

A *shortest paths distance matrix* $D: \mathcal{V} \times \mathcal{V} \to \mathbb{R}^+$ provides the minimum sum of edge distances between any two vertices in $\mathcal{V}$ without crossing any hull in $\mathcal{L}_\mathcal{U}$. Each edge cost $d_{loc(i),loc(j)} \in D$ (henceforth d_{ij}) is between two vertices, $i, j \in \mathcal{V}, i \neq j$. If there exists an unobstructed path between $loc(i)$ and $loc(j)$ (which does not go through any obstacle hull) the distance

is Euclidean $\|loc(i) - loc(j)\|$. If obstacles must be bypassed to go from $loc(i)$ to $loc(j)$, however, the distance is a sum of Euclidean distances following the shortest path between them (without crossing obstacles). The Floyd-Marshall graph algorithm is used to compute these shortest paths and distances exactly [13].

The set of vertices, including origin and destination vertex, that have to be visited to pick a batch is defined as $\mathcal{V}_b = \{v_s\} \cup v(b) \cup \{v_d\}, b \in \mathcal{B}$. A function can then be built which provides *the sequence of vertex visits in a batch TSP solution*:

$$T^b: \mathcal{V}_b \to \{v_i\}_{i=1}^{n}, n = |\mathcal{V}_b|, \tag{1}$$

$$T(b)_i = \begin{cases} v_s & i = 1 \\ v_k & 1 < i < n \\ v_d & i = n \end{cases} \tag{2}$$

where $v_k \in v(b)$ and i gives the sequence of visits. *The distance of a batch TSP solution* is similarly provided in a function:

$$D^b: T(b)_i \to \mathbb{R}^+, i \in \mathbb{N}^+, i \leq |T(b)|. \tag{3}$$

$$D(b) = \sum d_{T(b)_i T(b)_j}, i, j \in \mathbb{N}^+, j = i + 1, i < |T(b)| \tag{4}$$

Note D^b could be renamed D^{T^b} to clarify that the distance of a batch is computed over a certain *path* to visit all the products in the batch. $\mathcal{V}, E, G$ and D are assumed to be produced in a digitization preprocessing step and the computational effort at this stage is assumed to not be included in subsequent OBP optimization. Out of $\mathcal{V}, E, G$ and D only D is needed as input for OBP optimization assuming vehicles are capable of finding the shortest path between any two locations on their own. $\mathcal{V}, E, G$ are also needed for directions on how to follow the shortest path, and if visualizations of edges are sought, both of which are arguably important in an industrial OBP optimization service. One example of a visualization of G and a small OBP optimization instance can be seen in Fig. 4 below:

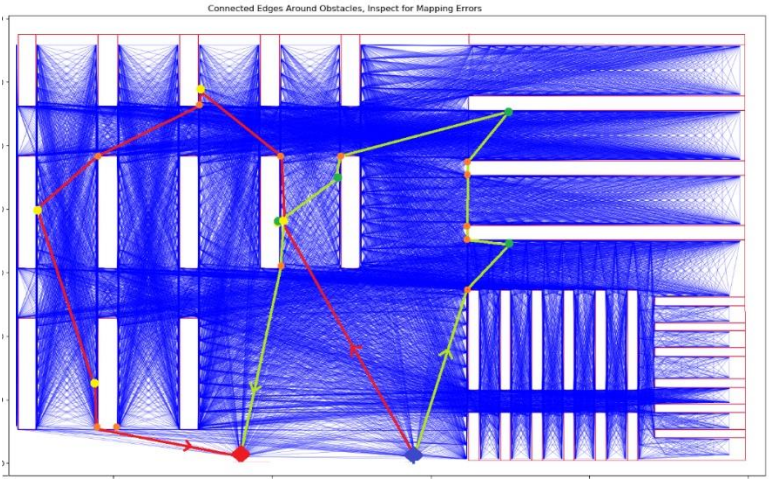

Fig. 4. Visualization of the digital graph (G) of a warehouse, and an example OBP with two orders, two vehicles and vehicle capacity of one order. Each blue line is an edge $e \in E$ that connects two vertices $(i, j), i, j \in \mathcal{V}$. The white hulls are racks (obstacles) laid out in an "unconventional" way and no edges pass through them. The orange vertices show a subset of $\mathcal{V}_{\mathcal{U}}$ and the green and yellow vertices along the racks are the sought products in $\mathcal{V}_{\mathcal{L}_\mathcal{P}}$ (where color indicates which order it belongs to). Note one of the products is visited by both vehicles. At the bottom the origin and destination, v_s and v_d can be seen (blue and red respectively). The OBP solution is here shown as the red and lime edges following the shortest paths between v_s, the yellow or green vertices and v_d (the two paths are obtained using T^b above).

3.2 General OBP formulation

A set-partitioning formulation with an exponential number of binary variables is used to formulate the layout-agnostic general OBP. The binary decision variable x_{mb} is used to indicate whether batch $b \in \mathcal{B}$ is assigned to vehicle $m \in \mathcal{M}$ ($x_{mb} = 1$, if m is assigned to b, $x_{mb} = 0$ otherwise). The binary decision variable x_{mo} is used to indicate whether order $o \in \mathcal{O}$ is assigned to vehicle $m \in \mathcal{M}$ ($x_{mo} = 1$ if m is assigned $o, x_{mo} = 0$ otherwise). The binary decision variable x_{ml} is used to indicate whether vehicle m visits location $l \in \mathcal{L}_\mathcal{P}$ ($x_{ml} = 1$ if m visits $l, x_{ml} = 0$ otherwise).

$$min \sum_{b \in \mathcal{B}} D(b) x_{mb}, m \in \mathcal{M} \tag{5}$$

$$\sum_{m \in \mathcal{M}} x_{mo} = 1, \forall o \in \mathcal{O} \tag{6}$$

$$\sum_{l \in loc(o)} x_{ml} \geq x_{mo}, \forall o \in \mathcal{O}, m \in \mathcal{M} \tag{7}$$

$$q(b) \leq q(m) x_{mb}, b \in \mathcal{B}, q \in \{w, vol, k\}, m \in \mathcal{M} \tag{8}$$

The optimization aim of the OBP (5) is to assign batches to vehicles such that the sum of the distances of all batches is minimized. (6) ensures that each unassigned order is assigned to exactly one vehicle (*order-integrity*). (7) ensures that every product location in every order assigned to a vehicle is visited at least once. This inequality is what renders the OBP a general *Steiner*-VRP. (8) ensures capacity of vehicles is never exceeded.

3.3. Single batch OBP formulation

The general OBP formulation is problematic to work with due to the large number of possible combinations of vehicles and batches. Below is a proposal for a more tractable problem where the aim is to find a batch for an already selected vehicle. After vehicle *m* has been selected the aim is to assign as many orders as possible to it while keeping batch distance at a minimum:

$$\underset{b \in \mathcal{B}}{argmin}\, D(b) \tag{9}$$

$$\exists q\big(q(b) + q(o) \geq q(m)\big), \forall o \in \mathcal{O}, o \notin b, q \in \{w, vol, k\} \tag{10}$$

where $k(o)$ (i.e. the number of orders in an order) is 1. The aim in the *single batch* OBP (9) is to, for a given vehicle m, find a single batch b with the minimal batch distance. Constraints (6), (7), and (8) from the general OBP still apply (for the given vehicle). Constraint (10) is further added to ensure that the number of orders in the batch is as large as possible (for all unassigned orders there exists a weight, volume or number of orders quantity which will exceed vehicle capacity if the order is added to the batch). Without this maximization of number of orders an optimization algorithm would always create a batch with just a single order because this would produce the minimal batch distance. The single batch OBP formulation is a specific version of the so called

minimum cost maximal knapsack packing problem (MCMKP) if distance is treated as "profit" and number of orders as knapsack "weight" (according to the definition by [30]). Note in the formulation here batch "weight" and "volume" are not included in the maximization since this would impose decision making over the importance of the different quantities (which one is most important to maximize while not exceeding vehicle capacity). The intention of the single batch OBP formulation is to provide the means with which to build an efficient *single batch* OBP optimization algorithm. This algorithm can then be used to produce one batch at a time within an algorithm which optimizes the general OBP, as proposed in Algorithm 1 below:

Algorithm 1. Iteration of single Batch OBP optimization.

1. total_OBP_cost ← 0
2. **WHILE** $\mathcal{O}$:
3. m ← select_vehicle($\mathcal{M}$)
4. b ← single_batch($\mathcal{O}, m, D$)
5. total_OBP_cost += $D(b)$
6. $\mathcal{O}$ ← remove_batched($\mathcal{O}, b$)

Algorithm 1 runs with the assumption that there are always enough vehicles to choose from, and it creates single batches until there are no more unassigned orders left. The total cost is expressed in the TSP path distances of the batches $D(b)$. After a batch has been created its orders are removed from $\mathcal{O}$.

4. Experimental results

This section evaluates the computational effort and memory requirement needed to generate the datastructures used by the formulation in Section 3. The only datastructure needed for OBP optimization is the distance matrix D, but graph G, including shortest paths between all locations are also included (Table 1).

Table 1. Experimental results for the digitization of distances and shortest paths.

Name	Number of locations	Number of polygons	Memory allocated		CPU time (s)
			Disk (mb)	RAM (mb)	
c6953_B01	781	0	14.4	78	541
c7561_B02	288	16	2.7	65	268
c0543_B03	760	1	14.2	79	501
c9109_AYD	218	5	0.7	62	195
c3495_BER	579	69	61	83	620
c0054_JUL	752	42	12.2	81	763
c3401_DAD	1384	296	377	385	5616
c9543_ARA	4037	234	520	574	26028
c2456_CAG	6491	306	1219	1290	65232

CPU used: Intel Core i7-4710MQ 2.5 GZ 4 cores, 8GB of RAM

Computational time and memory requirement grows fast with number of locations in the digitization procedure. The largest instance included 6491 defined locations and required 18 hours of *CPU-time*. Please note the computation only has to be run once (and re-run if the obstacle layout is changed in the warehouse). Once the graph has been generated, distances and shortest paths can be queried quickly by pre-allocating them in Random Access Memory (RAM), which is why RAM usage is also a relevant parameter. "Number of locations", denoted as $|\mathcal{L}|$ in Section 3, and the number of products in each defined location, varies depending on precision sought in the digitization process. For example, the warehouse denoted c9543_ARA, holds around 40000 products, but there are only 4037 defined locations. Each location in that case represents the products within an area of around 3 m^2 on the horizontal axis and 5 shelf levels on the vertical axis, with a total of around 10 products represented by every defined location. Clearly, a faster digitization process would be achieved if more products were mapped to the same locations, but then the digital model would be less precise. The tradeoff between memory and CPU-time on the one hand, and digitization precision on the other, is an interesting topic left for future work.

5. Conclusion

This paper set out to formulate an Order Batching Problem (OBP) that does not depend on the way in which racks or other obstacles are laid out in the warehouse. A digitization procedure to generate necessary datastructures was first described. A minisum set-partitioning formulation with an exponential number of binary variables was introduced for the layout-agnostic OBP. A more tractable version of the OBP, the *single batch* OBP, was additionally formulated where the aim is to find a single batch for an already specified vehicle. Experiments evaluating CPU-times and memory footprints for generating necessary datastructures was presented. In ensuing work new layout agnostic OBP optimization algorithms and benchmark instances will be introduced.

References

[1] R. de Koster, T. Le-Duc, and K. J. Roodbergen, 'Design and control of warehouse order picking: A literature review', *European Journal of Operational Research*, vol. 182, no. 2, pp. 481–501, 2007, doi: https://doi.org/10.1016/j.ejor.2006.07.009.

[2] X. Jiang, Y. Zhou, Y. Zhang, L. Sun, and X. Hu, 'Order batching and sequencing problem under the pick-and-sort strategy in online supermarkets', *Procedia Computer Science*, vol. 126, pp. 1985–1993, 2018, doi: https://doi.org/10.1016/j.procs.2018.07.254.

[3] G. P. Sharp and D. R. Gibson, 'Order batching procedures', *58*, no. European Journal of Operational Research, 1992.

[4] N. Gademann and V. de S. Velde, 'Order batching to minimize total travel time in a parallel-aisle warehouse', *IIE Transactions*, vol. 37, no. 1, pp. 63–75, 2005.

[5] C. A. Valle and B. A. Beasley, 'Order batching using an approximation for the distance travelled by pickers', no. European Journal of Operational Research, 2019.

[6] H. Ratliff and A. Rosenthal, 'Order-Picking in a Rectangular Warehouse: A Solvable Case of the Traveling Salesman Problem', *Operations Research*, vol. 31, pp. 507–521, 1983.

[7] J.-F. Cordeau, G. Laporte, M. Savelsbergh, and D. Vigo, ‘Vehicle Routing’, in *Transportation, handbooks in operations research and management science*, vol. 14, 2007, pp. 195–224.
[8] Y. A. Bozer and J. W. Kile, ‘Order batching in walk-and-pick order picking systems’, *International Journal of Production Research*, vol. 46, no. 7, pp. 1887–1909, 2008, doi: 10.1080/00207540600920850.
[9] O. Briant, H. Cambazard, D. Cattaruzza, N. Catusse, A.-L. Ladier, and M. Ogier, ‘An efficient and general approach for the joint order batching and picker routing problem’, *European Journal of Operational Research*, vol. 285, no. 2, pp. 497–512, 2020, doi: https://doi.org/10.1016/j.ejor.2020.01.059.
[10] O. Kulak, Y. Sahin, and M. E. Taner, ‘Joint order batching and picker routing in single and multiple-cross-aisle warehouses using cluster-based tabu search algorithms’, *Flexible Services and Manufacturing Journal*, vol. 24, no. 1, pp. 52–80, Mar. 2012, doi: 10.1007/s10696-011-9101-8.
[11] M. Masae, C. H. Glock, and E. H. Grosse, ‘Order picker routing in warehouses: A systematic literature review’, *International Journal of Production Economics*, vol. 224, p. 107564, 2020, doi: https://doi.org/10.1016/j.ijpe.2019.107564.
[12] P.-Y. Chang, P. Damodaran *, and S. Melouk, ‘Minimizing makespan on parallel batch processing machines’, *International Journal of Production Research*, vol. 42, no. 19, pp. 4211–4220, 2004, doi: 10.1080/00207540410001711863.
[13] L. J. van Rensburg, ‘Artificial intelligence for warehouse picking optimization - an NP-hard problem’, M.Sc., Uppsala University, 2019.
[14] C. A. Valle, J. E. Beasley, and A. S. da Cunha, ‘Optimally solving the joint order batching and picker routing problem’, *European Journal of Operational Research*, vol. 262, no. 3, pp. 817–834, Nov. 2017, doi: 10.1016/j.ejor.2017.03.069.
[15] J. Li, R. Huang, and J. B. Dai, ‘Joint optimisation of order batching and picker routing in the online retailer’s warehouse in China’, *International Journal of Production Research*, vol. 55, no. 2, pp. 447–461, 2017, doi: 10.1080/00207543.2016.1187313.
[16] H. Psaraftis, M. Wen, and C. Kontovas, ‘Dynamic Vehicle Routing Problems: Three Decades and Counting’, *Networks*, vol. 67, 2015, doi: 10.1002/net.21628.
[17] M. Bué, D. Cattaruzza, M. Ogier, and F. Semet, ‘A Two-Phase Approach for an Integrated Order Batching and Picker Routing Problem’, 2019, pp. 3–18.

[18] M. Bortolini, M. Faccio, E. Ferrari, M. Gamberi, and F. Pilati, ‘Design of diagonal cross-aisle warehouses with class-based storage assignment strategy’, *The International Journal of Advanced Manufacturing Technology*, vol. 100, no. 9, pp. 2521–2536, Feb. 2019, doi: 10.1007/s00170-018-2833-9.
[19] A. Fumi, L. Scarabotti, and M. Schiraldi, ‘The Effect of Slot-Code Optimization in Warehouse Order Picking’, *International Journal of Business and Management*, vol. 5, 2013, doi: 10.5772/56803.
[20] K. R. Gue and R. D. Meller, ‘Aisle configurations for unit-load warehouses’, *IIE Transactions*, vol. 41, no. 3, pp. 171–182, 2009, doi: 10.1080/07408170802112726.
[21] S. Henn, ‘Algorithms for on-line order batching in an order picking warehouse’, *Computers & Operations Research*, vol. 39, no. 11, pp. 2549–2563, 2012, doi: https://doi.org/10.1016/j.cor.2011.12.019.
[22] M.-C. Chen and H.-P. Wu, ‘An association-based clustering approach to order batching considering customer demand patterns’, *Omega*, vol. 33, no. 4, pp. 333–343, 2005, doi: https://doi.org/10.1016/j.omega.2004.05.003.
[23] S. Henn and G. Wäscher, ‘Tabu search heuristics for the order batching problem in manual order picking systems’, *European Journal of Operational Research*, vol. 222, no. 3, pp. 484–494, 2012, doi: https://doi.org/10.1016/j.ejor.2012.05.049.
[24] Ç. Cergibozan and A. Tasan, ‘Genetic algorithm based approaches to solve the order batching problem and a case study in a distribution center’, *Journal of Intelligent Manufacturing*, pp. 1–13, 2020, doi: 10.1007/s10845-020-01653-3.
[25] K. Braekers, K. Ramaekers, and I. V. Nieuwenhuyse, ‘The vehicle routing problem: State of the art classification and review’, *Computers & Industrial Engineering*, vol. 99, pp. 300–313, 2016, doi: https://doi.org/10.1016/j.cie.2015.12.007.
[26] V. Pillac, M. Gendreau, C. Guéret, and A. L. Medaglia, ‘A review of dynamic vehicle routing problems’, *European Journal of Operational Research*, vol. 225, no. 1, pp. 1–11, 2013, doi: https://doi.org/10.1016/j.ejor.2012.08.015.
[27] M. Mansouri, F. Lagriffoul, and F. Pecora, ‘Multi Vehicle Routing with Nonholonomic Constraints and Dense Dynamic Obstacles’, 2017, doi: 10.1109/IROS.2017.8206195.
[28] D. D. Bochtis and C. G. Sørensen, ‘The vehicle routing problem in field logistics part I’, *Biosystems Engineering*, vol. 104, no. 4, pp. 447–457, 2009.

[29] R. D. Santis, R. Montanari, G. Vignali, and E. Bottani, ‘An adapted ant colony optimization algorithm for the minimization of the travel distance of pickers in manual warehouses’, *European Journal of Operational Research*, vol. 267, no. 1, pp. 120–137, 2018.
[30] F. Furini, I. Ljubić, and M. Sinnl, ‘An effective dynamic programming algorithm for the minimum-cost maximal knapsack packing problem’, *European Journal of Operational Research*, vol. 262, no. 2, pp. 438–448, 2017, doi: https://doi.org/10.1016/j.ejor.2017.03.061.

Layout-Agnostic Order-Batching Optimization*

Johan Oxenstierna[1,2][0000−0002−6608−9621], Jacek Malec[1][0000−0002−2121−1937], and Volker Krueger[1][0000−0002−8836−8816]

[1] Dept. of Computer Science, Lund University, Box 118, SE-221 00 LUND
{johan.oxenstierna,jacek.malec,volker.krueger}@cs.lth.se
https://rss.cs.lth.se/
[2] Kairos Logic AB, Lund, Sweden

Abstract. Order-batching is an important methodology in warehouse material handling. This paper addresses three identified shortcomings in the current literature on order-batching optimization. The first concerns the overly large dependence on conventional warehouse layouts. The second is a lack of proposed optimization methods capable of producing approximate solutions in minimal computational time. The third is a scarcity of benchmark datasets, which are necessary for data-driven performance evaluation. This paper introduces an optimization algorithm, SBI, capable of generating reasonably strong solutions to order-batching problems for any warehouse layout at great speed. On an existing benchmark dataset for a conventional layout, Foodmart, results show that the algorithm on average used 6.9% computational time and 105.8% travel cost relative to the state of the art. New benchmark instances and proposed solutions for various layouts and problem settings were shared on a public repository.

Keywords: order-batching Problem · Order Picking · Discrete optimization

1 INTRODUCTION

There are many optimizable processes within warehouse operations. One of these is *order-picking*, which refers to the retrieval of shipment orders, where each order contains one or several products (items stored in the warehouse) [23]. As much as 55% of all operating expenses in a warehouse are allocated for order-picking [21]. A common method with which to conduct order-picking is *order-batching*, where each picker (vehicle) is set to pick a batch of one or more orders [37]. Within optimization literature order-batching is known as the Order-Batching Problem (OBP) [15] or the Joint Order-Batching and Picker Routing Problem (JOBPRP) [39]. The Picker Routing Problem is the Traveling

* Supported by the Wallenberg Autonomous Systems Program.

Salesman Problem (TSP) [33] applied in warehouses (henceforth the Picker Routing Problem is referred to as TSP). Most of the literature assesses quality of batches based on travel cost estimation while still calling the problem an OBP (without incorporating *picker routing* in the term), and this paper follows this convention. The OBP is usually formulated as a special version of the more well known Vehicle Routing Problem (VRP) [13]. While the general objective in the OBP is the same as in the VRP, i.e., to assign a set of vehicles to visit a set of locations at minimum travel cost, the literature on the OBP includes two distinguishing features:

- *Order-integrity*: In the OBP products of one order cannot be picked by more than one vehicle [18] whereas in the VRP this constraint is not used (orders are not defined in the VRP) [13].
- *Obstacle-layout*: As far as we are aware, all previous work on the OBP requires a certain form of obstacle layout in the warehouse (the *conventional layout*) (e.g. [5,24,38]). The conventional layout means that warehouse racks are placed in Manhattan style blocks with parallel aisles and cross-aisles (see Figure 1 a). The VRP does not have this requirement.

We are not aware of any reference in the literature which suggests a proportion of conventional versus unconventional layouts in the warehousing domain. Figure 1 includes examples of unconventional layouts used in industry. We see an overly large reliance on conventional layouts as a shortcoming in research on OBP optimization.

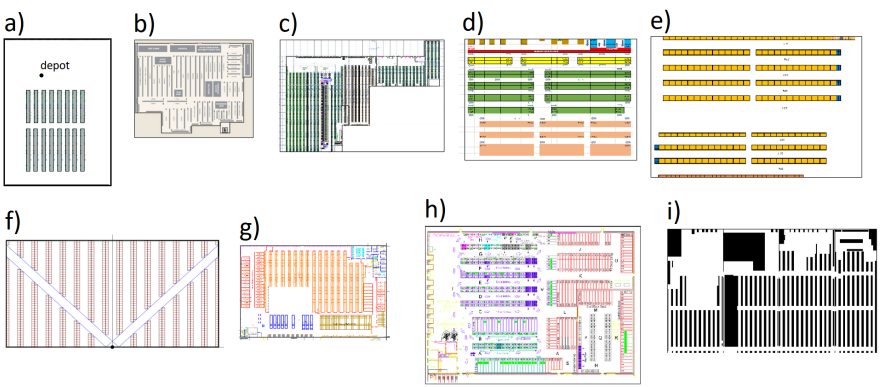

Fig. 1: Examples of warehouse layouts. All except a) are unconventional.

A second identified shortcoming concerns the subject of an OBP optimization module's required computational time, versus the ease with which it can be

integrated with a Warehouse Management System (WMS). The WMS manages the overall operation of a warehouse and there is a complex interaction between processes such as order-picking, delivery scheduling, quality assurance, location tracking, packing, verification, shipping, replenishment, yard management, labor management etc., and time margins are usually tight [4]. The WMS gets orders for picking dynamically during the workday. If a subset of these orders are sent to an optimization module, which will select some of them to be picked by a vehicle, it is therefore preferable, from an optimization point of view, to have this selection and corresponding picking tour computed before new orders have arrived to the warehouse. The simplest form of integration is by a synchronous request/response cycle between the WMS and the optimization module, instead of an asynchronous setup where the WMS first sends a request for optimization followed by the collection of a response at a later time (when the original request may already be obsolete). Synchronous request/response is only possible if optimization can be completed within a few seconds. This paper showcases the kind of OBP optimization performance achievable in such a short time.

A third identified shortcoming is a scarcity of publicly shared benchmark datasets on the OBP. These types of datasets are crucial to allow for experiment reproducibility and peer collaboration.

Our contributions are as follows:

1. The introduction of an optimization algorithm, SingleBatchIterated (SBI), with capability of producing fast approximate solutions to the OBP, irrespective of warehouse layout. SBI's performance is evaluated against the state of the art on Foodmart, a publicly available dataset, which models a warehouse with a conventional layout. The evaluation concerns distance minimization as well as computational time.
2. The introduction of a publicly shared OBP dataset with six types of warehouse layouts and 203 test-instances. Optimization results using SBI and various settings are included in each instance.

2 LITERATURE REVIEW

OBP's for warehouses with conventional layouts have been formulated using integer programming (e.g. [39]) or set-partitioning (e.g. [15]). The conventional layout appears in these formulations as required input parameters such as "number of aisles" [5], "intra-aisle distance" [7], "the cross-distance between two consecutive aisles" [15] or "number of vertices in the subaisle" [39]. Some authors have requested investigations into more layouts than the conventional layout [14, 16, 18, 27]. One benefit of this generalization is that more problem scenarios within logistics could be explored as OBP's. One drawback is that it

is very challenging to reduce the OBP solution space without taking advantage of regularities in the layout [39].

Authors often discuss OBP optimization with regard to two fundamental components: 1. Order to vehicle assignments. 2. Solving the TSP's needed to visit all products in the proposed order assignments. The two components can either be optimized jointly [39] or in separate phases [3,38]. The TSP component is often optimized using linear time S-shape or Largest-gap heuristics [18, 35] which are specifically designed for conventional layouts. The order to vehicle assignment component is often optimized using so called *proximity batching*, which heuristically ensures vehicles are assigned orders whose pick products are located close together [15]. Sharp & Gibson [37] propose First-Come-First-Served (FCFS), Space Filling Curve (SFC) and Sequential Minimal Distance (SMD) heuristics to ensure closeness between batched products. Rosenwein [36] proposes Minimum Additional Aisle (MAA) and Centre of Gravity (COG) heuristics. Ho et al. [20] propose 25 different heuristics to initialize and then add one order at a time to a batch until vehicle capacity is exceeded. These heuristics are sometimes collectively referred to as *seed heuristics* or *seed algorithms* [22]. Another heuristic optimization method for order to vehicle assignment is the so called Clark & Wright (C&W) savings algorithm [5]. In this algorithm travel cost to pick individual orders are first estimated and then compared against the cost required to pick larger collections of orders. This algorithm is known to produce batches with less travel cost than seed algorithms, while the computational effort is 100-200 times greater [22].

The OBP optimization objective can be stated as minimizing the sum of all TSP solution costs needed to pick all products (henceforth referred to as *minisum*) [5,6] or to minimize the maximum TSP solution costs (*minimax*) [18]. Solution cost is mostly expressed in terms of distance or time. The latter is more complex but also more realistic to work with as it involves predicting vehicle velocities, time to search for and pick items on shelves etc.

There is a broad array of focus areas in the literature on OBP optimization, reflecting different types of warehouse models and constraints. Chew & Tang [11], for example, examine the relationship between the travel cost of a vehicle, number of available vehicles and where products are stored in the warehouse. The latter is an optimization problem on its own called the Storage Location Assignment Problem (SLAP) [9]. It is rarely studied in conjunction with batching although there is a clear interdependence [18, 29]. If there are different origin and destination locations for vehicles the OBP is said to be a multi-depot or Dial-A-Ride Problem (DARP) [18]. A basic multi-depot example is whenever vehicles are set to drop off their picked orders at one pre-designated location, then move to another pre-designated location to collect empty boxes, i.e. orders that have not been picked yet, before moving out to collect a new batch. If all products that need to be picked are assumed to be known apriori the OBP is said to be *static* or *off-line* as opposed to *dynamic* or *on-line* [40]. Proposed op-

timization programs for the OBP versions described above include integer and mixed integer [5], dynamic programming [6], data mining [10], clustering [24] and meta-heuristics such as Tabu Search [19], Ant Colony Optimization [26] and Genetic Algorithms [8].

Computational time used for OBP optimization and its relevance within warehouse operations is a topic only discussed on a high level if at all in the literature. Some authors set timeouts for optimization but these are only arbitrarily defined to simulate a "tolerable" time horizon [24]. The largest test-instance results with 30 - 5000 orders in Briant et al. [6], were achieved after optimization was set to run between 30 minutes to 2 hours. Briant et al., do not discuss whether a WMS provider would be interested in allocating 30 minutes for generating 6 optimized batches out of 30 unassigned orders.

There exist two OBP benchmark datasets: Foodmart [39] and HappyChic [6] which are designed for static OBP's and two conventional layouts. The vast majority of benchmarking in OBP research is not carried out on public datasets, but instead on a described model/simulation of a warehouse with a conventional layout. For comparison, in the related research domain on the Vehicle Routing Problem (VRP), there are several widely used benchmark datasets which researchers use to evaluate optimization performance, including the Solomon, Christofides, Taillard, Augerat et al., Fisher and Kilby instances [28, 30–32]. A commonly used data format for VRP instances is TSPLIB [17]. We have extended on TSPLIB to introduce new OBP instances in the experimental part of this paper (Section 5).

3 PRELIMINARIES

In this section we define all relevant terms and parameters that will be needed for the reminder of the paper. For better readability, we keep the definitions on an intuitive level and use mathematical precision only where necessary.

A batch b is defined as a set of orders from customers, selected out of a set of unassigned orders. The unassigned orders are denoted O and the set of all possible batches is denoted B. Each order contains a set of products and each product has a volume and weight. A batch is picked by a vehicle, m, selected out of a set of available vehicles, $m \in M$. The vehicle's capacities are expressed in number of orders, weight and volume. Each product has a location in the warehouse. The union of all locations in a batch b is retrievable with a function loc^b. The sequence of location visits a vehicle follows to pick a batch (including an origin and a destination location for the vehicle) is computable with a function T^b. Note T^b gives a solution to a Traveling Salesman Problem (TSP). The distance of T^b is computable with a function D^b. The D^b function makes use of a distance matrix which contains the shortest distance between all locations in a given warehouse (without crossing obstacles). The distance

matrix is assumed pre-computed. For a presentation of the digitization steps followed to produce it see [34].

The optimization objective of the minisum OBP [5,15,19] is to assign batches to vehicles such that the distance required to pick the orders is minimized, while not breaking any of the following constraints:

1. Each unassigned order is assigned to exactly one vehicle (order-integrity).
2. Each product location in each order assigned to a vehicle must be visited at least once.
3. Capacities of vehicles may never be exceeded.

The proposed optimization algorithm (Section 4) makes use of an optimization module which optimizes a more tractable form of the OBP, the so called *single batch* OBP. The optimization objective of the single batch OBP is to find a single batch b with the minimal batch distance. Constraints 2 and 3 still apply for the single batch version of the problem. The following additional constraint is added:

4. The number of orders in a single batch must be as large as possible.

Without this last constraint i.e. the maximization of number of orders, a single batch optimization algorithm would always create a batch with just a single order. This is because the minimal batch distance is always achieved if the batch is made up of just a single order. Note that it is possible to define constraint 4 as both a constraint and an objective. Constraint 4 is delimited from including weight and volume of products in the maximization since this would necessitate decision making over whether weigth, volume or number of orders is more important. Both above models are concerned with *static* OBP's (Section 2) i.e. ones where all unassigned orders can be batched at any time.

4 OPTIMIZATION ALGORITHM

In this section we will introduce the SingleBatchIterated (SBI) optimization algorithm, which produces an approximate solution to the minisum OBP. Internally it makes use of the *SingleBatch* algorithm, which produces an approximate solution to the single batch OBP (Section 3). SingleBatch is shown in the lower rectangular box in Fig 2. It is used to produce single batches and corresponding picking tours iteratively until there are no more unassigned orders left.
A vehicle is first selected from a set of vehicles (a) and a subset of unassigned orders from the set of all unassigned orders is selected (b). This subset selection is done to reduce the amount of computational time needed for the subsequent optimization. A single batch b as well as a TSP solution for that batch are then computed using the SingleBatch optimization algorithm (c). The distance of

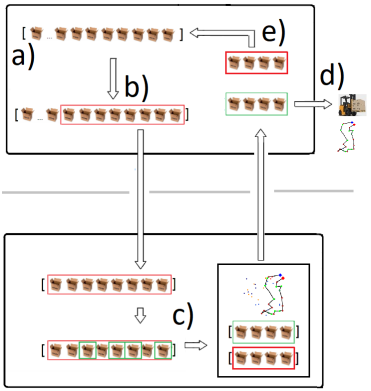

Fig. 2: Flowchart showing SingleBatchIterated (SBI). First unassigned orders are processed according to priority and a subset of orders is then sent to the SingleBatch optimization algorithm (lower box), which produces a single batch and the TSP solution required to pick that batch. The algorithm runs until all unassigned orders have been batched.

the TSP solution is added to the total cost of the OBP solution (initialized as 0). The selected vehicle is dispatched to pick batch b (d). The orders in b are removed from the set of unassigned orders (e). The steps in Fig 2 correspond to Algorithm 1 shown below.

Algorithm 1: SBI

cost $\leftarrow 0$
while O **do**
  $m \leftarrow$ select_vehicle(M)
  $O_s \leftarrow$ select_subset(O)
  $b \leftarrow$ single_batch(O_s, m, D)
  cost = cost $+D(b)$
end

Algorithm 2: SingleBatch

`single_batch`(O_s, m, D)
// Phase 1
$b_{ord} \leftarrow$ seed_algorithm(O_s, m, D)
// Phase 2
$b_{tour} \leftarrow$ solve_tsp(b, D)
return b

The SingleBatch algorithm, Algorithm 2, takes a subset of unassigned orders O_s, a vehicle m and the distance matrix D as input. Order selection using one of two seed algorithms is used to initialize a batch and assign orders to it (b_{ord}) until vehicle capacity runs out. A tour to pick the batch (b_{tour}) is computed using the Concorde TSP solver (details for Concorde are beyond the scope of this paper, for details see [2, 12]). The SingleBatch function returns the batch (including the orders and the tour).

SBI requires that there are enough vehicles to batch all orders. This delimitation is used because the vehicle selection part is handled by the Warehouse Management System (WMS) in the intended industrial application (the WMS takes over the full handling of the upper rectangle in Fig 2, i.e., it decides when and which vehicles should be assigned a batch).

The purpose of the SingleBatch seed algorithm (inside Algorithm 2) is to return a batch of orders that allows subsequent TSP optimization (for the locations in that batch) to result in a short distance. One way to achieve a batch selection quickly is to use heuristics such as Sequential Minimal Distance

(SMD) [37] or Centre of Gravity (COG) [36], and these are tested and compared in the experimental part of this paper. SMD and COG can be used to output a scalar value ("distance") that estimates the distance that would be achieved if the locations of the products of two orders were used to formulate and optimize a TSP. The "seed algorithm" works sequentially by adding an order at a time to a sequence of assigned orders (i.e. the single batch). The "seed order" denotes the order which was last added to the sequence (while the sequence is being populated). SMD and COG are used to search for an unassigned order, with a low "distance" to the seed order, to add next. The first order in the sequence can for example be selected randomly [37]. In SingleBatch's seed algorithm it is instead selected as the order with the least sequential minimal distance (SMD) or the shortest distance to the centre of gravity (COG) (depending on which is used). To enable this the vehicle origin location is used as a first seed placeholder. Using SMD or COG for the first order selection is motivated by the SingleBatch optimization objective which states that the distance of the batch should be as short as possible regardless of how many orders end up in that batch.

SMD is computed using the following:

$$SMD(s,o) = \sum_{i \in s} \min_{j \in o} |d_{ij}|, \quad o \in O, o \notin b, s \in b \tag{1}$$

where d_{ij} is the distance between product i in order s (the seed) and product j in unassigned order o. $SMD(s,o)$ is then calculated as the sum of these minimal distances d_{ij}. Sharp & Gibson [37] present a way in which to compute d_{ij} in the conventional layout scenario. For the unconventional layout scenario it is given as $d_{ij} \in D$ (D is the shortest paths distance matrix, assumed pre-computed).

The COG heuristic was introduced by Rosenwein [36] and is for a single order given as:

$$COG(o) = \frac{1}{|o|} \sum_{p \in o} a_p \tag{2}$$

where a_p denotes the location of the product, and $|o|$ is the number of products in the order. The COG of two orders is given by the Manhattan distance between two order COG's: $COG(s,o) = |COG(s) - COG(o)|$ where s and o denote the seed order and an unassigned order, respectively. Note this version of COG does not make use of distance matrix D and hence does not take the warehouse layout into account.

Once the order with the least SMD or COG has been found it is added to the batch and set as the new seed. New orders are then added in the same way until vehicle capacity is full or there are no more unassigned orders left.

5 EXPERIMENTS

In this section we first discuss the datasets used i.e. Foodmart and the new test instances generated. Then,

1. we discuss OBP results using our SBI approach on the datasets in terms of distance minimization, as well as computational times.
2. we compare results using a seed algorithm running either the SMD or COG heuristics.
3. we compare computational times required by the seed algorithm and the TSP solver.

5.1 Datasets

Foodmart Foodmart contains test-instances for static OBP's and a conventional layout. It was introduced by Valle et al. [39] and includes 135 test-instances with up to 50 unassigned orders and 7 larger testing-instances with 50 to 5000 orders. The layout has 3 cross-aisles and a maximum of 8 aisles (see Fig. 1 a)). There is only a single origin and destination location.

In Foodmart each vehicle carries 8 bins, where each bin has a volume capacity of "40 V". Each product has a volume ranging from 1 to 40 V, and if an order contains products whose sum of V's exceeds 40, or exceeds the volume left in any of the 8 bins, the order may be split between different bins on the same vehicle. This way to formulate vehicle capacity is specific to Foodmart. There are many possible alternatives, e.g. maximum number of orders [25], products [5], volume [8] or weight [24]). The number of available vehicles is unlimited in Foodmart.

Presented results for Foodmart in [6,39] include optimal OBP results for 130 test-instances where the number of orders to be batched varies between 5 - 100. These instances can therefore be used to evaluate our approach against optimal results on conventional layouts. We believe the gap between SBI's results and optimal results can be used as an estimate of how far away from optimality SBI's results are on unconventional layouts.

Generated test-instances Six different types of warehouse layouts on a 80×80 grid were first generated with the following name-tags: "No obstacles", "conventional layout with 3 cross-aisles and 12 aisles", "1 single rack", "12 racks", "NR1" and "NR2" (see Fig 4). "NR" stands for non-regular. The unconventional layouts were chosen as simplified representations of real examples seen in the industry (see Figure 1).

Using the generated layouts, 203 test-instances on a modified TSPLIB format were then generated (30-40 instances for each layout) [3]. The modifications

[3] https://github.com/johanoxenstierna/OBP_instances

made to the TSPLIB are described in a text file in the provided link. For simplicity vehicle capacity is the same for all vehicles and only expressed in number of orders (between 2-30) in these instances, and experiments involving more capacity types (e.g. volume, weight, number of products, Foodmart type bins and combinations of capacities and/or vehicle types) are left for future work. The number of vehicles in the instances is set as the ceiling of number of unassigned orders divided by the vehicle number of orders capacity (denoted k^M): $|M| = \lceil \frac{|O|}{k^M} \rceil$. Concerning where the products are placed in the warehouse (see Section 2 for an explanation for why this is relevant in OBP's), either 1, 2 or 4 rectangular storage assignment zones are used. These zones are placed anywhere on the grid and are generated in two steps: First a random x, y storage zone centroid coordinate within the 80×80 grid is generated. Then storage locations for products (for each order in the generated instance) are generated such that the Manhattan distance between the product location and the storage zone centroid coordinate do not exceed a specified distance [4]. Each of the six layout types has a differing origin and destination location where vehicles start and end their tours.

5.2 Experimental Results

Since the vehicles in Foodmart use bins into which orders are placed the SingleBatch algorithm was first adapted to be able to handle that particular capacity type. To be exact, the modification was conducted within the call to the "seed_algorithm" function in Algorithm 2. Three modifications were made: 1. The batch object b was modified to include a key-value dictionary "bins" with 8 enumerated keys and corresponding values to keep track of how much volume has been taken up in each bin. 2. A function *check_candidate_order*(b, o) (inside "seed_algorithm"), which checks if a candidate order can be added to a batch without breaking constraints, was modified to find the bin which, if the order is added to it, comes as close as possible to the 40 volume capacity without breaking it. 3. If there exists such a bin its key is returned, the order is added to the batch and the given bin is updated with the added volume. If the order cannot be added to any bin in this batch it is excluded and added to a different batch at a later stage. Only SMD was used as order selection heuristic for the Foodmart experiment.

The OBP experimental results are summarized in Table 1 (Foodmart) and Table 2 (generated instances). On the Foodmart instances an average of 105.8% distance and 6.9% computational time was achieved relative to reported optimal results in [6,38]. The result shows that fast approximate OBP optimization can be accomplished with a relatively small penalty in added distance.

[4] it is is called "min_distance_to_slotting_zone" and can be found in a specs JSON in each instance.

Concerning the comparison of the SMD and COG heuristics, results only concern the generated instances (since COG was not used on Foodmart). It was found that, on the 203 generated OBP instances, SBI with SMD yielded solutions with 97.9% distance and 131% computational time, relative to SBI with COG. Within SingleBatch, the seed algorithm on average used only 7.3% of the total computational time versus the TSP solver Concorde's 92.7%. On average, the seed algorithm requires 0.05 — 0.1 seconds to construct a batch using SMD, whereas Concorde requires anywhere between 0.001 – 3 seconds to solve a batch TSP, depending on various factors such as number of product locations in the batch (see Fig. **??**).

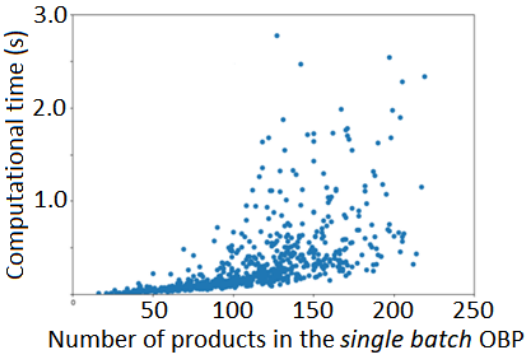

Fig. 3: CPU-time (y-axis) of the SingleBatch algorithm versus number of products in the single batch OBP's (x-axis) (this figure excludes results on Foodmart and "NR1").

No attempt was made to infer how features such as layout, storage zones and depot locations affect the computational times shown in Fig. **??**. Concorde has a high degree of internal variance when it comes to computational time [1,2,12]. It would therefore require a large number of OBP test instances to make this type of inference.

6 CONCLUSION

This paper introduced an optimization algorithm, SingleBatchIterated (SBI), capable of producing strong approximate solutions to the OBP at minimal computational time for both conventional and unconventional warehouse layouts. The algorithm was evaluated on the Foodmart benchmark dataset, where it showed that OBP solutions could be obtained at great speed and with a relatively low penalty in added distance compared to optimal results. Additionally, a new OBP dataset with several types of layouts, depot locations and storage zone settings was introduced. Proposed solutions using SBI were uploaded together with visualizations of the new instances.

The vast majority of computational time in SBI was allocated to TSP solving rather than order selection. Results show that this is mostly due to the TSP solver Concorde, which has a high internal variance in terms of computational time. Instead of replacing Concorde with a TSP optimizer which is more stable with regard to computational time, it is deemed more relevant to allocate more computational time at the order selection phase. As Fig. **??** and Table 2 show, most OBP instances were optimized in well under 1 second, which allows for more optimization in many scenarios. One alternative could be to add the *savings algorithm* (Section 2) as an alternative for order selection and to use it if there are relatively few products in the batch. Further work on dataset generation is also needed, especially for OBP instances involving dynamicity and more vehicle capacity options.

References

1. Applegate, D., Cook, W., Dash, S., Rohe, A.: Solution of a Min-Max Vehicle Routing Problem. INFORMS Journal on Computing **14**, 132–143 (2002)
2. Applegate, D.L., Bixby, R.E., Chvatal, V., Cook, W.J.: The traveling salesman problem: a computational study. Princeton university press (2006)
3. Azadnia, A., Taheri, S., Ghadimi, P., Samanm, M., Wong, K.: Order Batching in Warehouses by Minimizing Total Tardiness: A Hybrid Approach of Weighted Association Rule Mining and Genetic Algorithms (Scientific World Journal) (2013)
4. Bartholdi, J., Hackman, S.: Warehouse and distribution science Release 0.98 (2019)
5. Bozer, Y.A., Kile, J.W.: Order batching in walk-and-pick order picking systems. International Journal of Production Research **46**(7), 1887–1909 (2008)
6. Briant, O., Cambazard, H., Cattaruzza, D., Catusse, N., Ladier, A.L., Ogier, M.: An efficient and general approach for the joint order batching and picker routing problem. European Journal of Operational Research **285**(2), 497 – 512 (2020)
7. Bué, M., Cattaruzza, D., Ogier, M., Semet, F.: A Two-Phase Approach for an Integrated Order Batching and Picker Routing Problem. pp. 3–18 (2019)
8. Cergibozan, C., Tasan, A.: Genetic algorithm based approaches to solve the order batching problem and a case study in a distribution center. Journal of Intelligent Manufacturing pp. 1–13 (2020)
9. Charris, E., Rojas-Reyes, J., Montoya-Torres, J.: The storage location assignment problem: A literature review. International Journal of Industrial Engineering Computations **10** (2018)
10. Chen, M.C., Wu, H.P.: An association-based clustering approach to order batching considering customer demand patterns. Omega **33**(4), 333 – 343 (2005)
11. Chew, E.P., Tang, L.C.: Travel time analysis for general item location assignment in a rectangular warehouse. European Journal of Operational Research **112**(3), 582 – 597 (1999)
12. Cook, W.: Concorde TSP Solver (2020), http://www.math.uwaterloo.ca/tsp/concorde/index.h
13. Cordeau, J.F., Laporte, G., Savelsbergh, M., Vigo, D.: Vehicle Routing. In: Transportation, handbooks in operations research and management science, vol. 14, pp. 195–224 (2007)

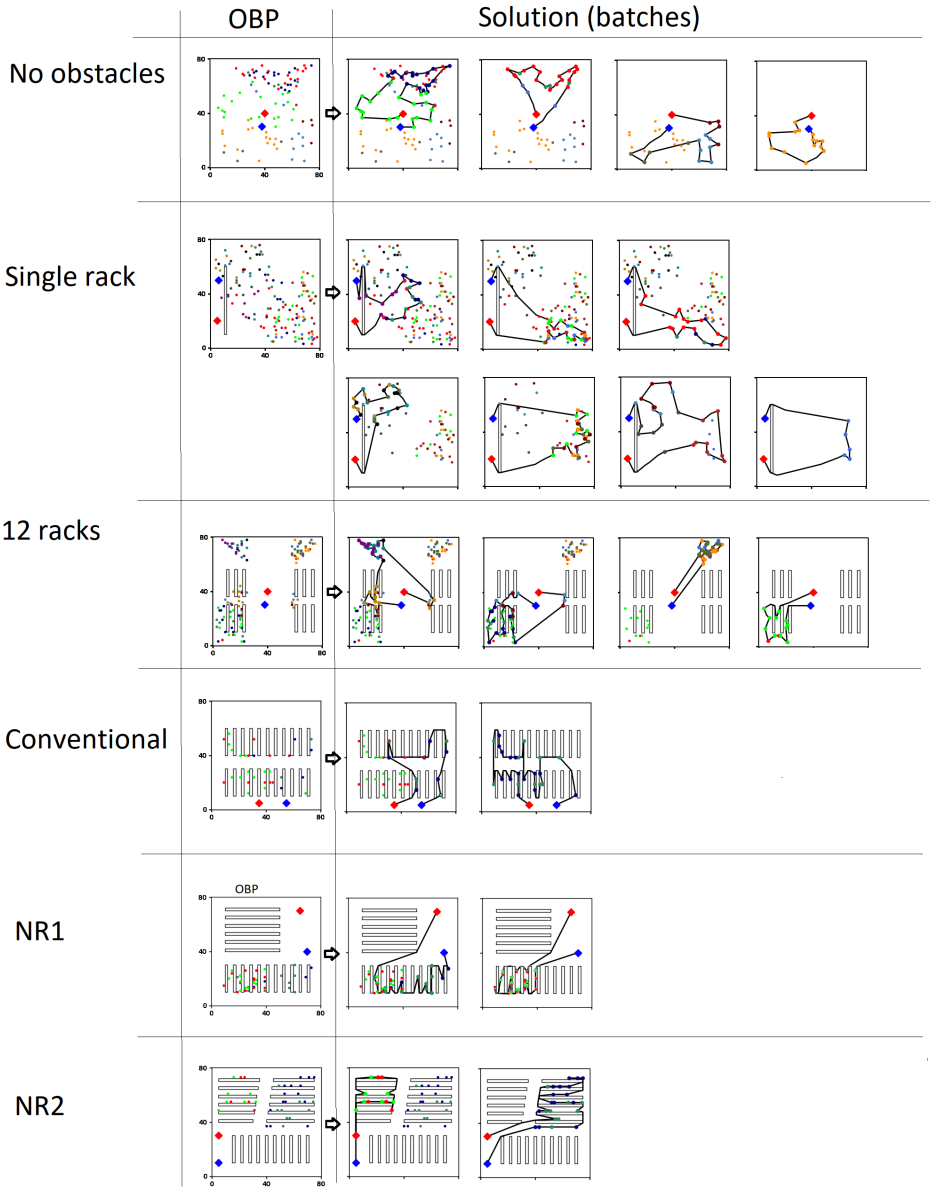

Fig. 4: Six examples of instances (one for each layout type) and solutions (from top left to bottom right) using SBI with the SMD heuristic. The larger red and blue dots are the origin and destination locations for vehicles. Each smaller dot denotes a product and their color denotes the order which the product belongs to.

Name	Number of products	Number of orders	Briant et al. (2020)		SBI		Relative distance	Relative CPU time
			Distance	CPU total time (s)	Distance	CPU total time (s)		
d5_ord5	59	5	348.59	1.15	348.59	0.21	1.000	0.183
d5_ord6	67	6	364.81	1.11	364.81	0.05	1.000	0.045
d5_ord7	74	7	374.81	1.84	374.81	0.14	1.000	0.076
d5_ord8	81	8	503.75	3.23	503.9	0.17	1.000	0.053
d5_ord9	88	9	539.61	4.27	561.5	0.18	1.041	0.042
d5_ord10	95	10	581.42	10.89	611.61	0.18	1.052	0.017
d5_ord11	102	11	611.63	53.67	645.6	0.18	1.056	0.003
d5_ord12	109	12	613.40	45.27	649.51	0.18	1.059	0.004
d5_ord13	116	13	623.39	75.86	685.8	0.19	1.100	0.003
d5_ord14	123	14	637.73	116.19	647.46	0.08	1.015	0.001
d5_ord15	130	15	653.42	166.54	663.46	0.1	1.015	0.001
d5_ord20	165	20	862.10	1596.8	960	0.13	1.114	0.000
d5_ord25	200	25	1 083.85	7246.5	1233.22	0.37	1.138	0.000
d5_ord30	230	30	1 000.53	7304.03	1208.61	0.28	1.208	0.000
d10_ord5	80	5	371.09	0.87	371.09	0.09	1.000	0.103
d10_ord6	91	6	377.09	1.23	377.09	0.2	1.000	0.163
d10_ord7	102	7	549.80	1.77	565.61	1.23	1.029	0.695
d10_ord8	113	8	584.18	1.43	648.18	0.42	1.110	0.294
d10_ord9	124	9	637.37	7.15	693.47	0.58	1.088	0.081
d10_ord10	134	10	661.80	16.23	685.8	1.2	1.036	0.074
d10_ord11	144	11	692.43	114.83	729.8	0.41	1.054	0.004
d10_ord12	154	12	703.42	182.96	752.18	0.7	1.069	0.004
d10_ord13	164	13	712.24	124.13	761.9	0.59	1.070	0.005
d10_ord14	174	14	723.83	182.6	761.8	0.48	1.052	0.003
d10_ord15	183	15	881.69	189.19	958.4	0.46	1.087	0.002
d10_ord20	223	20	976.32	955.79	1064.61	3.62	1.090	0.004
d10_ord25	258	25	1 187.30	2677.58	1355.42	3.66	1.142	0.001
d10_ord30	293	30	1 159.58	7305.76	1295.8	3.71	1.117	0.001
d20_ord5	91	5	573.77	0.66	573.77	0.33	1.000	0.500
d20_ord6	105	6	656.18	1.19	678.18	0.16	1.034	0.134
d20_ord7	119	7	689.80	8	714.18	0.2	1.035	0.025
d20_ord8	132	8	697.80	10.54	698.18	0.94	1.001	0.089
d20_ord9	145	9	726.80	41.18	751.8	1.13	1.034	0.027
d20_ord10	158	10	905.26	37.92	964.86	0.51	1.066	0.013
d20_ord11	171	11	980.51	11.65	1009.27	0.28	1.029	0.024
d20_ord12	183	12	999.25	41.92	1056.84	0.74	1.058	0.018
d20_ord13	195	13	1 008.12	82.72	1040.89	0.73	1.033	0.009
d20_ord14	207	14	1 011.08	70.6	1084.99	3.56	1.073	0.050
d20_ord15	219	15	1 025.46	87.81	1096.99	3.57	1.070	0.041
d20_ord20	279	20	1 332.26	619.56	1435.98	0.51	1.078	0.001
d20_ord25	334	25	1 602.86	3288.5	1738.87	0.76	1.085	0.000
d20_ord30	389	30	1 843.13	7271.35	2025.12	2.74	1.099	0.000

CPU used: Intel Core i7-4710MQ 2.5 GZ 4 cores, 8GB of memory

Table 1: This table shows experimental results on a subset of the Foodmart dataset

Name	Number of products avg.	Layout	Storage zones	SMD avg. distance	SMD avg. CPU time (s)	COG avg. distance	COG avg. CPU time (s)
s1	88	None	1	516	0.30	516	0.33
s2	118	None	2	380	0.41	380	0.39
s2_1o	111	1 rack	2	369	0.65	369	0.58
s2_12o	110	12 racks	2	377	0.43	380	0.39
s2_12o40	42	12 racks	2	290	0.03	293	0.12
s2_12o120	122	12 racks	2	421	0.59	422	0.55
s2_12o200	117	12 racks	2	395	0.55	409	0.49
s1_3CAo	66	Conv.	1	618	0.24	621	0.32
s4	83	None	4	840	0.47	852	0.38
s2_3CAo	131	Conv.	2	652	0.56	683	0.42
s2_NR1o	254	NR1	2	1233	1.88	1295	1.47
s2_NR1o	146	NR1	2	1001	0.91	1015	0.72
s2_NR2o	140	NR2	2	650	0.93	651	0.68
Foodmart	173	Conv.	-	903	0.91	-	-

CPU used: Intel Core i7-4710MQ 2.5 GZ 4 cores, 8GB of memory

Table 2: This table summarizes the experimental results on 14 types of instances (Foodmart can be seen in the lowest row).

14. Fumi, A., Scarabotti, L., Schiraldi, M.: The Effect of Slot-Code Optimization in Warehouse Order Picking. International Journal of Business and Management **5** (2013)
15. Gademann, N., Velde, V.d.S.: Order batching to minimize total travel time in a parallel-aisle warehouse. IIE Transactions **37**(1), 63–75 (2005)
16. Gue, K.R., Meller, R.D.: Aisle configurations for unit-load warehouses. IIE Transactions **41**(3), 171–182 (2009)
17. Hahsler, M., Kurt, H.: TSP – Infrastructure for the Traveling Salesperson Problem. Journal of Statistical Software **2**, 1–21 (2007)
18. Henn, S.: Algorithms for on-line order batching in an order picking warehouse. Computers & Operations Research **39**(11), 2549 – 2563 (2012)
19. Henn, S., Wäscher, G.: Tabu search heuristics for the order batching problem in manual order picking systems. European Journal of Operational Research **222**(3), 484 – 494 (2012)
20. Ho, Y.C., Su, T.S., Shi, Z.B.: Order-batching methods for an order-picking warehouse with two cross aisles. Computers & Industrial Engineering **55**(2), 321 – 347 (2008)
21. Jiang, X., Zhou, Y., Zhang, Y., Sun, L., Hu, X.: Order batching and sequencing problem under the pick-and-sort strategy in online supermarkets. Procedia Computer Science **126**, 1985 – 1993 (2018)
22. Koster, M.B.M.D., Poort, E.S.V.d., Wolters, M.: Efficient orderbatching methods in warehouses. International Journal of Production Research **37**(7), 1479–1504 (1999)
23. Koster, R.d., Le-Duc, T., Roodbergen, K.J.: Design and control of warehouse order picking: A literature review. European Journal of Operational Research **182**(2), 481 – 501 (2007)

24. Kulak, O., Sahin, Y., Taner, M.E.: Joint order batching and picker routing in single and multiple-cross-aisle warehouses using cluster-based tabu search algorithms. Flexible Services and Manufacturing Journal **24**(1), 52–80 (2012)
25. Le-Duc, T., Koster, R.M.B.M.d.: Travel time estimation and order batching in a 2-block warehouse. European Journal of Operational Research **176**(1), 374–388 (2007)
26. Li, J., Huang, R., Dai, J.B.: Joint optimisation of order batching and picker routing in the online retailer's warehouse in China. International Journal of Production Research **55**(2) (2017)
27. Masae, M., Glock, C.H., Grosse, E.H.: Order picker routing in warehouses: A systematic literature review. International Journal of Production Economics **224**, 107564 (2020)
28. Mańdziuk, J., Świechowski, M.: UCT in Capacitated Vehicle Routing Problem with traffic jams. Information Sciences **406-407**, 42 – 56 (2017)
29. Nieuwenhuyse, I., De Koster, R., Colpaert, J.: Order batching in multi-server pick-and-sort warehouses. Katholieke Universiteit Leuven, Open Access publications from Katholieke Universiteit Leuven (2007)
30. Okulewicz, M., Mańdziuk, J.: The impact of particular components of the PSO-based algorithm solving the Dynamic Vehicle Routing Problem. Applied Soft Computing **58**, 586 – 604 (2017)
31. Pillac, V., Gendreau, M., Guéret, C., Medaglia, A.L.: A review of dynamic vehicle routing problems. European Journal of Operational Research **225**(1), 1 – 11 (2013)
32. Psaraftis, H., Wen, M., Kontovas, C.: Dynamic Vehicle Routing Problems: Three Decades and Counting. Networks **67** (2015)
33. Ratliff, H., Rosenthal, A.: Order-Picking in a Rectangular Warehouse: A Solvable Case of the Traveling Salesman Problem. Operations Research **31**, 507–521 (1983)
34. Rensburg, L.J.v.: Artificial intelligence for warehouse picking optimization - an NP-hard problem. Master's thesis, Uppsala University (2019)
35. Roodbergen, K.J., Koster, R.: Routing methods for warehouses with multiple cross aisles. International Journal of Production Research **39**(9), 1865–1883 (2001)
36. Rosenwein, M.B.: A comparison of heuristics for the problem of batching orders for warehouse selection. International Journal of Production Research **34**, 657–664 (1996)
37. Sharp, G., Gibson, D.: Order batching procedures. European Journal of Operational Research (58) (1992)
38. Valle, C., Beasley, B.: Order batching using an approximation for the distance travelled by pickers (European Journal of Operational Research) (2019)
39. Valle, C., Beasley, J.E., da Cunha, A.S.: Optimally solving the joint order batching and picker routing problem. European Journal of Operational Research **262**(3), 817–834 (2017)
40. Yu, M., Koster, R.B.M.d.: The impact of order batching and picking area zoning on order picking system performance. European Journal of Operational Research **198**(2), 480 – 490 (2009)

Efficient Order Batching Optimization using Seed heuristics and the Metropolis algorithm

Johan Oxenstierna, Jacek Malec and Volker Krueger

Abstract

Order Picking in warehouses is often optimized using a method known as Order Batching, which means that one vehicle can be assigned to pick a batch of several orders at a time. There exists a rich body of research on Order Batching Problem (OBP) optimization, but one area which demands more attention is computational efficiency, especially for optimization scenarios where warehouses have unconventional layouts and vehicle capacity configurations. Due to the NP-hard nature of the OBP, computational cost for optimally solving large instances is often prohibitive. In this paper we compare the performance of two approximate optimizers designed for maximum computational efficiency. The first optimizer, Single Batch Iterated (SBI), is based on a Seed Algorithm, and the second, Metropolis Batch Sampling (MBS), is based on a Metropolis algorithm. Trade-offs in memory and CPU-usage and generalizability of both algorithms is analysed and discussed. Existing benchmark datasets are used to evaluate the optimizers on various scenarios. On smaller instances we find that both optimizers come within a few percentage points of optimality at minimal CPU-time. For larger instances we find that solution improvement continues throughout the allotted time but at a rate which is difficult to justify in many operational scenarios. SBI generally

outperforms MBS and this is mainly attributed to the large search space and the latter's failure to efficiently cover it. The relevance of the results within Industry 4.0 era warehouse operations is discussed.

1. Introduction

Order Picking is the process in which sets of products (orders) are retrieved from locations in a warehouse. *Order Batching* is a method in which vehicles can be assigned to pick several orders at a time. Order Batching can be formulated as an optimization problem known as the Order Batching Problem (OBP) (Gademann et al., 2001) or the Joint Order Batching and Picker Router Problem (JOBPRP) (Valle et al., 2017), where the Picker Router Problem is a Traveling Salesman Problem (TSP) applied in a warehouse environment (Ratliff & Rosenthal, 1983). We consider the OBP and JOBPRP versions equivalent if solutions to the OBP are assumed to include TSP solutions (henceforth we use the term OBP to refer to this version). There are several other versions and focus areas in OBP's, including dynamicity, traffic congestion, depot setups and obstacle layouts. One focus area is optimization aimed towards maximum computational efficiency. As will be laid out in Section 2, computational efficiency has both direct and indirect impacts on the quality of warehouse operations. Authors generally consider it to be important, but there are significant differences in how CPU-times and timeouts are used. Although the variability of OBP versions and corresponding results concerning computational efficiency is high, we believe more research in this domain is warranted. We delimit our work to OBP's where the objective is to minimize aggregate distances, and as measurement of computational efficiency we use the rate with which aggregate distance is reduced through CPU-time. We use the following two OBP optimizers: *Single Batch Iterated* (SBI) (Oxenstierna et al., 2021, 2022) and *Metropolis Batch Sampling* (MBS) (introduced in this paper). We compare the aggregate distance result between the two optimizers, and also compare against results on public OBP benchmark datasets. We use a distance based OBP cost because this is the predominant Key Performance Indicator (KPI) in benchmark datasets. Although a KPI based on capital cost is what is mostly sought by warehouse management, it is more complex: There are a multitude of features that can go into capital, such as time-based aspects of work, traffic congestion, maintenance, ergonomics etc. A distance based

KPI allows for a simpler model and a more generalized way in which to reproduce benchmark data and results.

We only work with CPU-times in the range 0 – 300 seconds. Results are compared with previous work by Aerts et al. (2021) and Henn & Wäscher (2012) who have proposed approximate optimization results for sets of smaller instance sizes. For smaller instances we also assess results against optimal results on the Foodmart dataset (Briant et al. 2020). For larger instances we use L09_251 (Oxenstierna et al., 2022). As far as we are aware, there exists no standard benchmark format in OBP research, rendering experiment reproducibility difficult. Further discussions on how to represent key OBP features in reproducible data is highly relevant. Our research contributions are as follows:

- An investigation into the importance of computational efficiency in OBP optimization.
- Experiments regarding computational efficiency of two OBP optimizers on existing test-instances.

2. Literature Review

In this section we first present how the OBP and some of its key features are formulated in the literature. Then we present commonly used OBP optimization algorithms and heuristics. Finally, we present how computational efficiency has been motivated and evaluated for different OBP models.

As several studies have pointed out, the Order Batching Problem (OBP) shares significant similarities with the more well-known Vehicle Routing Problem (VRP) (Cordeau et al., 2007; Valle et al., 2017; Valle & Beasley, 2019). Aerts et al. (2021) distinguish three points of separation between the OBP and a common VRP:

1. *Order-integrity* constraint: In the OBP, products belonging to an order may only be picked by one vehicle, whereas there exists no concept of orders or order-integrity in the common VRP.

2. *Number of visits* constraint: In the OBP the same location may be visited several times by various vehicles, whereas a location may only be visited once in the common VRP.
3. *Obstacle-layout*: In the OBP it is assumed that there exists an obstacle layout, whereas there is no such assumption in the common VRP.

Concerning the latter point, most of the research on the OBP assumes that the warehouse uses a *conventional* layout, which means racks are arranged with parallel aisles (between racks) and parallel cross-aisles (between sections of racks) (Masae et al., 2020). If these conditions are not met the layout is *unconventional* (see Figure 1).

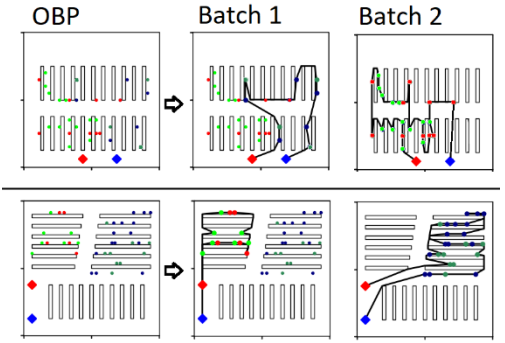

Figure 1: Examples of the conventional (top) and a unconventional (bottom) layout warehouse, and OBP's with four orders from Oxenstierna et al., (2022). The colored diamonds denote origin and destination locations. The colored dots denote products and the orders which they belong to. In the solutions (right of the arrows), one vehicle is assigned to pick the red and lime orders and a second vehicle is assigned to pick the blue and green orders.

Aerts et al. argue that the OBP can be modelled as a *Clustered* VRP (CluVRP) *with weak cluster constraints*. Weak cluster constraints mean that a vehicle may visit the locations in several clusters of locations in any sequence. The CluVRP was first introduced by Defryn & Sörensen (2017) and according to Aerts et al. (2021) it is equivalent to the OBP since clusters can be mapped as orders. In experiments they utilize this problem on a conventional layout warehouse and on OBP scenarios involving up to 100 orders.

For conventional layouts, proposed optimization algorithms include integer programming (Valle et al., 2017), clustering (Kulak et al., 2012), datamining (Chen & Wu, 2005), dynamic programming (Briant et al., 2020) meta-heuristics and heuristics. Examples of meta-heuristics include Variable Neighborhood Search (Aerts et al., 2021), Tabu Search (Henn & Wäscher, 2012), Ant Colony Optimization (Li et al., 2017) and Genetic Algorithms

(Cergibozan & Tasan, 2020). The heuristic algorithms can be divided into three categories: Priority rule-based algorithms, savings algorithms and seed algorithms (Henn et al., 2010). Priority-rule based algorithms build batches by sorting orders according to a heuristic, for example First-Come-First-Serve, First-Fit or Best-Fit. In savings algorithms batches with single orders are first initialized and evaluated. Then, pairs, triplets and larger batches of orders are constructed and the combination with the best total result is retrieved (Henn & Wäscher, 2012). In seed algorithms batches are generated in two phases: Seed-selection and order addition. In the first phase an initial seed order is selected. In order-addition orders are then added to the seed order. There are many choices for seed algorithms, with corresponding advantages and disadvantages depending on the usecase (Ho et al., 2008; Kulak et al., 2012; Scholz et al., 2017). One example is the Sequential Minimal Distance (SMD) heuristic (Sharp & Gibson, 1992), where the sum of minimal distances between products in the seed order and remaining orders is computed:

$$SMD(s,o) = \sum_{i \in s} \min_{j \in o} |d_{ij}|,\ o \in \mathcal{O}, o \notin b, s \in b \qquad (1)$$

where s denotes a seed order in batch b, where o denotes an order which does not exist in b, and where i and j denote products in order s and o respectively.

Whenever there are more than two products in a batch, some form of TSP optimization is often used within the OBP optimizer. For conventional layouts, the highly efficient *S-shape* or *Largest Gap* algorithms are commonly used (Henn, 2012; Roodbergen & Koster, 2001). We are not aware of any attempts to extend these to unconventional layouts. Given a distance matrix is provided, however, TSP's can be optimized reasonably fast using OR-tools (Kruk, 2018) or Concorde (D. Applegate et al., 2002; D. L. Applegate et al., 2006). Concorde, for example, is almost guaranteed to find the shortest path of a TSP with 100 nodes in less than one second (D. L. Applegate et al., 2006).

OBP models can be divided into static and dynamic: Dynamic models are generally more realistic than static ones (incoming orders are there assumed to be known beforehand). The literature still tends to model OBP's as static since dynamicity incurs more complexity (Scholz et al., 2017).

The importance of computational efficiency in OBP optimization can be derived from two types of factors. The first type has an immediate impact on operations in the warehouse. As an example, optimization should ideally be faster than the time it takes a vehicle to finish a picking round (Henn, 2012;

Scholz et al., 2017). Otherwise, vehicles must wait in an idle state at the depot while optimization finishes.

The second type concerns a larger perspective with flexible 4.0 industry-era integration and business utility. As an example, if an OBP optimization module is deployed on the cloud as a 3rd party software service (*SaaS*), a Warehouse Management System (WMS) client may be more interested in buying it if it is safe and simple to integrate. Longer CPU-times generally make it harder to set up a system (at least as a microservice) so that these two conditions are fulfilled (Esposito et al., 2016). Furthermore, rental and electricity cost of servers can naturally be assumed to rise with CPU-times (Naumenko & Petrenko, 2021).

In the broader literature on the OBP, the second type of factors are rarely discussed. CPU times are chosen to be "tolerable" (Kulak et al., 2012), "reasonable" (Bozer & Kile, 2008) or "acceptable" (Aerts et al., 2021; Scholz et al., 2017), but often lack in concrete explanations of what these terms entail. Some examples are provided below for how researchers have used CPU-times and timeouts in optimization experiments with OBP's.

For approximate optimization, Henn & Wäscher (2012) use timeouts between 1 – 180 seconds for a heuristic optimizer and OBP's where 40 – 100 unassigned orders are to be batched. Aerts et al. (2021), use timeouts between 1 and 60 seconds on the same instance set and propose a meta-heuristic algorithm specifically designed to terminate at around 60 seconds, since solution improvement is found to be insignificant beyond that point. Both Aerts et al. and Henn & Wäscher's algorithms come to within 5% of the best solution overall within the first 10% of optimization time. Scholz et al. (2017) experiment with instances of similar size but in a dynamic setting and report a much lower efficiency: 70% of maximum allowed CPU-time is necessary to reach within 5% of best solution overall. Efficiency also decreases non-linearly with instance size in their results: For 10 orders their optimizer needs 2 seconds, for 100 orders it needs 11 minutes, and for 200 orders 60 minutes. Henn (2012) also presents an algorithm for dynamic OBP's and sets it to self-terminate after 60 seconds, partly due to operational considerations (to avoid vehicles from idling at the depot). Many publications do not present concrete results for timeouts or rate of solution improvement, or a low number of experiments (Azadnia et al., 2013; Bué et al., 2019; Jiang et al., 2018). Kulak et al. (2012) and Li et al. (2017), for example, present highly efficient meta-heuristic optimizers, but only on 5 to 10 instances, and do not include rate of solution improvement in their results. For authors presenting algorithms capable of finding optimal solutions to static OBP's, Henn & Wäscher (2012)

set timeouts between 2 – 1328 seconds for instances with up to 60 orders. Gademann et al. (2001), set timeouts to 10 – 30 minutes for up to 100 orders. Valle et al. (2017) and Briant et al. (2020), on the Foodmart dataset, present timeouts in the range 300 seconds to 2 hours to obtain optimal results for 20-45 orders.

These examples show that computational efficiency in OBP optimization is difficult to judge generally. Choice of static or dynamic modelling, optimal versus approximative optimization, experimental setup, instance sizes and the technology level of used software and hardware, are all aspects that can have a complex effect on results in this regard.

3. Problem Formulation

We define the OBP objective as the assignment of batches to vehicles such that the aggregate distance needed to pick the batches is minimized. Each batch b consists of a set of one or several orders $b \in 2^{\mathcal{O}}$, where each $o \in \mathcal{O}$ is a subset of products $o \in 2^{\mathcal{P}}$. Each product $p \in \mathcal{P}$ is a set which includes a unique product identifier, an order identifier, weight w and volume vol, $w, vol \in \mathbb{R}^+$. The sum of weight, volume or number of orders in a batch can be retrieved with function $q(b), q \in w, vol, k$. The x, y location coordinates of all products is defined as set $\mathcal{L}_{\mathcal{P}}$, and the location of a product is retrievable with function $l(p)$. The locations of the products in an order are retrievable with function $l(o) = \cup_{p \in o}\, l(p)$, and all locations in a batch are retrievable with function $l(b) = \cup_{o \in b}\, l(o)$. We define a single origin location for all vehicles l_s, a single destination location l_d and a set of polygonal obstacle location sets $\mathcal{L}_{\mathcal{U}}$. The aggregate of all locations is $\mathcal{L} = \{l_s\} \cup \{l_d\} \cup \mathcal{L}_{\mathcal{P}} \cup \mathcal{L}_{\mathcal{U}}$.

We build undirected graph $G = (V, E)$. Each vertex in V represents a unique location in $\mathcal{L}$ and function $v(l)$ gives a vertex for a location. The vertices in batch b includes the origin and destination vertices $v(b) = v(l_s) \cup v\big(l(b)\big) \cup v(l_d)$. E represents the set of all Euclidean edges between all locations that circumvent obstacles in $\mathcal{L}_{\mathcal{U}}$. Distance matrix D and shortest paths between all edges is computed using the Floyd-Warshall algorithm. How E and shortest paths can be constructed with polygonal obstacles is beyond the scope of this paper; for details see (Rensburg, 2019). We also permit a surjective relationship of products to locations, i.e., several types of products can be stored at the same location and the location represents several real locations in

the warehouse. This can be useful to help reduce the memory footprint of G. The path to pick batch b is retrievable with the following function:

$$T(b) = \{v_i\}_{i=1}^{n}, n = |v(b)|, \tag{2}$$

$$v_i = \begin{cases} v_s & i = 1 \\ v_k & 1 < i < n \\ v_d & i = n \end{cases} \tag{3}$$

and represents the solution to a Traveling Salesman Problem (TSP). The distance of $T(b)$ is retrievable with function $D(b) = \sum d_{T(b)_i T(b)_j}, i, j \in \mathbb{Z}^+, j = i + 1, i < |T(b)|$, where $d \in \mathbb{R}^+$ represents scalar entries in distance matrix D. Vehicles are defined as $m \in \mathcal{M}$ where each vehicle has capacities expressed in weight w, volume vol and number of orders k. The scenario where a vehicle m is assigned a batch, order, and/or product location is defined with binary variables x_{mb}, x_{mo} and x_{ml}, respectively. We then use the following OBP formulation:

$$min \sum_{b \in \mathcal{B}} D(b) x_{mb}, m \in \mathcal{M} \tag{4}$$

s.t.

$$\sum_{m \in \mathcal{M}} x_{mo} = 1, \forall o \in \mathcal{O} \tag{5}$$

$$\sum_{l \in loc(o)} x_{ml} \geq x_{mo}, \forall o \in \mathcal{O}, m \in \mathcal{M} \tag{6}$$

$$q(b) \leq q(m) x_{mb}, b \in \mathcal{B}, \quad q \in w, vol, k, m \in \mathcal{M} \tag{7}$$

where (4) states the objective, i.e., minimize distances for all generated batches $\mathcal{B}$, where (5) enforces order-integrity, where (6) enforces all locations in all orders to be visited at least once and where (7) ensures vehicle capacities are never exceeded. Since this OBP is highly intractable we also use a less ambitious objective in the *single batch* OBP:

$$\underset{b \in \mathcal{B}}{argmin}\, D(b) \tag{8}$$

where the aim is to find a single batch for an already selected vehicle. For this case we also enforce the single batch to come as close as possible to vehicle capacity: $\exists q\big(q(b) + q(o) \geq q(m)\big), \forall o \in \mathcal{O}, o \notin b, q \in w, vol, k.$

4. Optimization Algorithms

4.1 Single Batch Iterated (SBI)

SBI (Algorithm 1) is a heuristic multi-phase optimizer. In the core of the algorithm unassigned orders $\mathcal{O}$ are iteratively sent as input to the *SMD* (Sequential Minimal Distance) function, together with distance matrix D, a randomly chosen available vehicle and a semi-stochastic seed order index. The *SMD* function builds a single batch b by first selecting a seed order according to the seed index and then adds orders to it according to minimal distances (Equation 1). Batch b is then removed from the set of unassigned orders and the procedure repeats until all orders have been batched into $\mathcal{B}$. An approximate solution to the OBP (Equation 4) is thus obtained by pre-selecting vehicles and approximately solving a *single batch* OBP for each vehicle (Equation 8).

Algorithm 1: *Single Batch Iterated (SBI)*

$y^* \leftarrow \infty$
$\hat{y} \leftarrow \infty$
for $i = 1, ..., N$ **do**
  $\mathcal{O}_s \leftarrow \mathcal{O}$
  $\mathcal{B} \leftarrow \{\}$
  while $\mathcal{O}_s$ **do**
    $m \leftarrow$ select_vehicle($\mathcal{M}$)
    $b \leftarrow \text{SMD}(\mathcal{O}_s, m, D, i)$
    $\mathcal{B} \leftarrow \mathcal{B} \cup b$
  end
  $y \leftarrow TSP(\mathcal{B}, D)$
  update_best$(y, \hat{y}, y^*)$
end

Number of iterations (N) is used here for brevity and in the implementation (Section 5) a time-based condition is used to stop the outer loop. The purpose of the i index is to reduce the probability that the same solution is obtained multiple times, and, if used, it can be set to $N = |\mathcal{O}|$, for example. The paths to visit all locations in batches $b \in \mathcal{B}$, $T(b)$ and their distance, $\sum_{b \in \mathcal{B}} D(b) x_{mb}$, is computed using the OR-tools TSP optimization suite[1] (in the *TSP* function). OR-tools is set to finish quickly by using a number-of-iterations parameter, which is set to grow linearly with number of vertices in the TSP. In the update_best function, the aggregate distance between the new OR-tools cost (y) is compared against the best OR-tools cost obtained so far ($\hat{y}$). If the new cost is lower, the TSP's are optimally solved using Concorde[2] and if this is better than the previous Concorde best, the result is stored as the new best in y^*.

Since the number of SMD computations between orders is approximately cubic to number of orders, $|\mathcal{O}| \sum(|\mathcal{O}| - i), i \in [|\mathcal{O}| - 1]$, we use an SMD order-order enumerated matrix which is populated through the optimization procedure: If SMD between two orders does not exist in the matrix, it is

[1] https://developers.google.com/optimization/routing/tsp, collected 13-09-2021.

[2] http://www.math.uwaterloo.ca/tsp/concorde/index.html, collected 16-09-2021.

computed and pushed to the matrix. Once the value is stored it is subsequently queried. Caching SMD's reduces number of calls to SMD from cubic to square, at an insignificant increase in memory usage (~25 megabytes for 5000 orders assuming 8 bits per cell in the matrix). It should be noted that this only works for an SMD algorithm where the seed is defined as a single order, which cannot provide more than a noisy estimate of the subsequent TSP solution distance for batches with more than two orders. We still deem pairwise order-order SMD caching suitable, since distance estimates are inaccurate even if SMD's for larger collections of orders are computed (TSP optimization is required for accurate estimates). Caching could also be used to store all generated single batches and their solved TSP's in a hash tree or equivalent, to prevent the same TSP to be optimized twice (*memoization*). We leave an implementation of this for future work.

One potential issue with SBI is its reliance on the SMD heuristic. Although SMD makes sure the distance between orders is always minimized for a given batch, the number of orders to select from decreases through the single-batch while-loop in Algorithm 1. Hence, the last batch which is created in the while-loop can be assumed to be of worse quality in terms of distance minimization relative to the first.

4.2 Metropolis Batch Sampling (MBS)

MBS is a heuristic multi-phase optimizer which uses distance matrix D, the Concorde TSP solver (in the *TSP* function below) and the SMD heuristic to compute distance between orders. The main difference between SBI and MBS is that the latter only uses the SMD function to produce an initial solution. A Metropolis algorithm (Mackay, 1998) is then used to improve on it using the following procedure:

Algorithm 2: Metropolis Batch Sampling

$\mathcal{B}_1 \leftarrow \{\}$
while $\mathcal{O}$ **do**
  $m \leftarrow$ select_vehicle$(\mathcal{M})$
  $b \leftarrow \text{SMD}(\mathcal{O}, m, D)$
  $\mathcal{B}_1 \leftarrow \mathcal{B}_1 \cup b$
end
$y_1 \leftarrow TSP(\mathcal{B}_1, D)$

for i = 1, ..., N **do**
  $\mathcal{B}_{i+1} \leftarrow$ new_sample$(\mathcal{B}_i)$
  $y_{i+1} \leftarrow TSP(\mathcal{B}_{i+1}, D)$
  $\alpha \leftarrow min(1, \gamma(y_i / y_{i+1}))$
  $u \leftarrow \mathcal{U}(0, 1)$ // random uniform
  if $u \leq \alpha$ **do**
    $\mathcal{B}_i \leftarrow \mathcal{B}_{i+1}$
    $y_i \leftarrow y_{i+1}$
end

The upper while-loop is equivalent to the one in Algorithm 1. The lower for-loop consists of a Metropolis algorithm where each new sample is drawn from a previous one. The function new_sample$(\mathcal{B}_i)$ uses the following stationary distribution to describe the probability for a given new sample:

$$q(\mathcal{B}_{i+1}|\mathcal{B}_i) = e^{-2CH_d(\mathcal{B}_i, \mathcal{B}_{i+1})^P} \tag{9}$$

where C and P are constants and where the $H_d(\mathcal{B}_i, \mathcal{B}_{i+1})$ function denotes the number of swapping operations needed to obtain $\mathcal{B}_{i+1}$ from $\mathcal{B}_i$. A swapping operation is defined as a switch of position of two orders in an enumerated set of batches. Since number of swaps to go from $\mathcal{B}_i$ to $\mathcal{B}_{i+1}$ is always equal to number of swaps to go from $\mathcal{B}_{i+1}$ to $\mathcal{B}_i$, the q distribution is symmetrical, i.e., $q(\mathcal{B}_{i+1}|\mathcal{B}_i) = q(\mathcal{B}_i|\mathcal{B}_{i+1})$. A swap is only permitted if vehicle weight and volume capacity constraints are not broken.

The TSP's of the batches in the new sample are then solved using Concorde (Section 2) in the *TSP* function, and the aggregate cost is stored in y_{i+1}. The accept probability α is computed based on the following balance condition (Tak et al., 2018): If $y_{i+1} < y_i$ the new sample is always accepted. If $y_{i+1} \geq y_i$ the sample may still be accepted if a uniform random value is less than α. α depends on the ratio y_i/y_{i+1}.

Contrary to SBI, the search space of MBS is guaranteed to include the global optimum, provided the sampling function can output any $\mathcal{B}$ that does not break constraints and enough computational time. This may just as well be a liability, however, since the search space may be too large for the algorithm to see optimization gains within reasonable time. We add bias parameter $\gamma \in \mathbb{R}^+$ to allow for experiments where the search space of the algorithm is more restricted. Without the use of γ, the probability is high that the algorithm steps away from the SBI local minimum $\mathcal{B}_1$ in the very first iteration (which is likely to happen if y_1/y_2 is close to one).

The best sample is assumed to be stored throughout the optimization procedure (sample storage is omitted from the pseudo-code). Just as with Algorithm 1, a number of iterations parameter $N \in \mathbb{Z}^+$ is shown as stopping condition in Algorithm 2 for brevity, but in the experiments in Section 5, a CPU-time condition is used instead. Establishing a suitable N for converge is possible by studying covariance of samples, but it is challenging in the OBP case: $\mathcal{B}_i$ is a set of orders where the orders may contain a variable number of products at variable locations. Heuristics would thus be needed to quantify covariance between two samples.

5. Experiments

5.1 Benchmark datasets

The publicly shared OBP datasets Foodmart[3], L6_203[4] and L09_251[5] are used for experimentation. Foodmart was introduced by Valle et al. (2017) and models a warehouse with a conventional layout and a maximum of 8 aisles and 3 cross-aisles. A feature in Foodmart is that vehicles carry bins and that vehicle capacity is expressed as a volume unit per bin. If an order cannot fit in a single bin, splitting it between different bins is permitted. SBI and MBS are not designed for this feature (it constitutes an extra bin packing problem within the OBP), so a greedy heuristic module is attached to the optimizers for the Foodmart experiment (for details see Oxenstierna et al., 2021).

L6_203 and L09_251 model scenarios for one conventional and up to six unconventional warehouse layouts and multiple depots. In these instances, vehicle capacity is expressed in number of orders and total number of orders generally range between 4-50. All but 6 Foodmart instances also fit within this range. For total number of orders in the range 50-1000 we use L09_251.

Number of orders only gives a rough idea of how much CPU-time might reasonably be needed to optimize an OBP instance. Number of products per order and vehicle capacities are further examples of features that have a considerable impact. To classify instances by size, we use the amount of computational time needed to obtain the SMD baseline solution: 0-2, 2-4, 4-7 or >7 seconds. The resulting number of instances for the four classes are as follows: 0-2 s: 335, 2-4 s: 179, 4-7 s: 91, >7 s: 56. The maximum permitted CPU-time for the 0-2 s instances is set to 20 seconds and 300 seconds for the remaining ones. For all our experiments we use Intel Core i7-4710MQ 2.5 GZ 4 cores, 16 GB RAM.

[3] https://pagesperso.g-scop.grenoble inp.fr/~cambazah/ batching/, collected 19-05-2022 (135 instances).

[4] https://github.com/johanoxenstierna/OBP_instances, collected 19-05-2022 (257 instances).

[5] https://github.com/johanoxenstierna/L09_251, collected 19-05-2022 (269 instances).

5.2 Experiment results

Aggregations of all results are presented in Figure 2, Figure 3, Table 1 and Table 2 (Appendix). In Figure 2, the relative improvement rates from the baseline are shown for the two optimizers and four instance size classes. The shades around the lines represent 95% confidence intervals.

In terms of rate of solution improvement, SBI performs stronger than MBS across all instance size classes, and the difference grows with instance size. The MBS results shown are for parameter values $C = 0.1$, $P = 1$, $\gamma = 10$ (Section 4.2), retrieved from early testing. A γ value of 1 (the standard Metropolis algorithm), yields weaker results.

Overall, the solution improvement rates for the two smaller instance classes (blue and orange) corroborate those of Henn & Wäscher (2012) and Aerts et al. (2021): Improvements are significant in the initial stage of optimization (1-4% improvement over baseline within the first 10% of optimization) and then taper off. In our case all instances with up to 100 orders require no more than 2 seconds to obtain the SBI baseline.

The Foodmart instances (all except 6) fit within the smallest class and there we compare our strongest results against optimal results in Briant et al. (2020): On average a gap to optimality of 2.3% is achieved after a maximum of 10 seconds. The average gap between the baseline solution and the best solution found is 3.2% on Foodmart. On generated instances in L6_203 the corresponding gap is 3.5%. For the larger two instance classes (>4 seconds to find a baseline solution), the pattern is similar, but more time is needed to reach the same percentage improvement over the baseline.

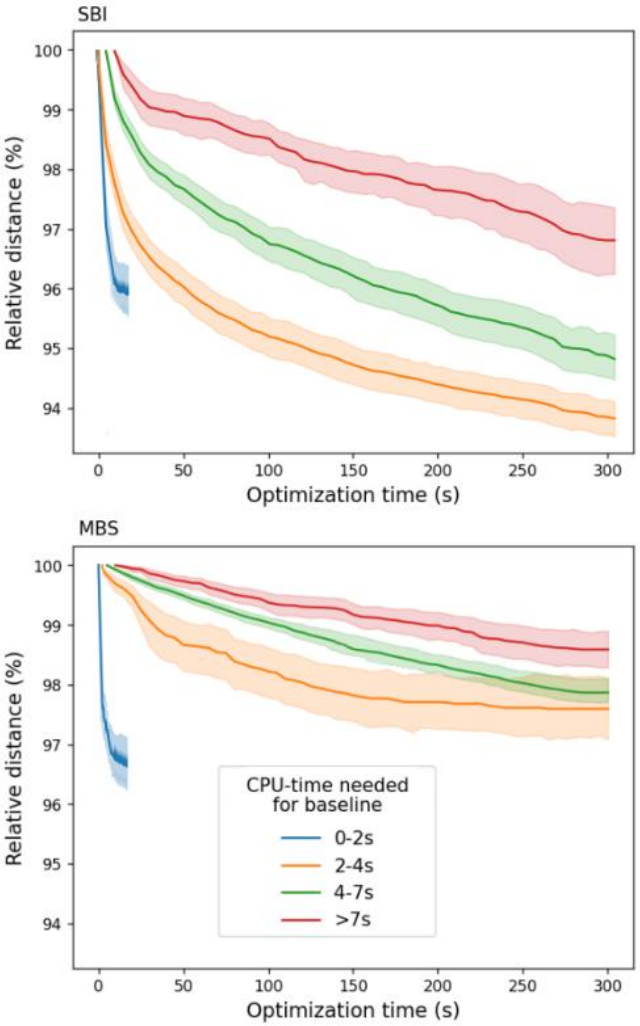

Figure 2: Optimization time versus relative OBP distances in percentages, for four instance size classes (661 total instances). The smallest instances (blue) end at a 20 second timeout.

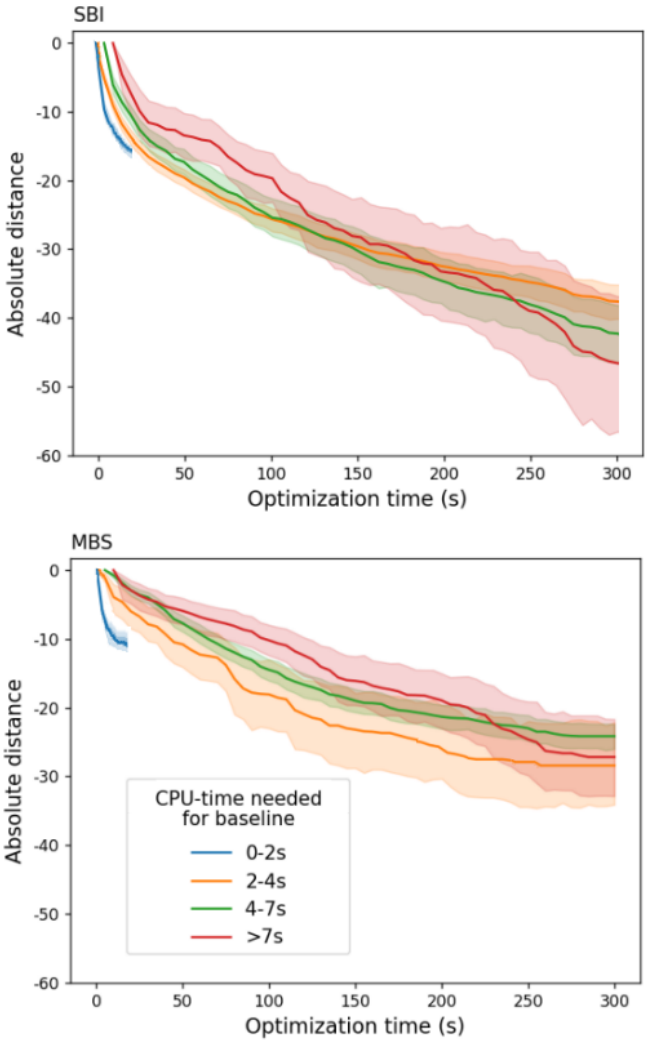

Figure 3: Optimization time versus standardized absolute distance savings, for four instance size classes.

In terms of absolute distance rate of improvement, we first standardize the data such that the average pick round is of similar length between the three datasets. The absolute distance improvements for the four instance size classes are shown in Figure 3. One observable pattern in Figure 3 is that larger OBP's tend to see more solution improvement. One explanation for this is that the

probability of finding a strong baseline decreases with larger instances, and possible optimization gains can therefore be assumed to also be larger. The red curves (>7 s), for example, yield weaker results in the beginning and stronger results in the end, relative to the green curves (4-7 s). It takes around 4 minutes to get there, however. Since there are only 56 instances in the red class, more data would be needed to investigate this pattern further and to narrow the confidence intervals.

The best achieved rates of solution improvement decrease to less than <1% / minute after the initial gains taper off (after 30 – 60 seconds for the larger instances in Figure 2). In terms of standardized distance, this is on average equivalent to around 18% of the length of a single batch TSP solution (~12 standardized distance units).

As discussed in Section 2, judgement of results in light of previous work is challenging due to the high variability of OBP models. Overall, we believe 1% / min is a slow rate of improvement and that it would be difficult to justify in many OBP scenarios, especially when considering the advantages of short CPU-times discussed in Section 2.

There are several possible reasons why MBS performs worse than SBI. One is that the relatively general application of the Metropolis algorithm faces a search space which is far too large for it to adequately sample within the allotted time. Even with a high amount of extra bias, imposed through manually added parameter γ, MBS is not able to find and improve on samples faster than SBI. The purely heuristic and more biased SBI optimizer has no global optimum guarantee due to the SMD heuristic, but it instead guarantees that each sample is a relatively strong local minimum. Using the semi-stochastic seed index within the SMD function (Section 4.1) also makes sure that the SBI local minimums are uncorrelated to some extent. Another possible advantage of SBI is that it uses an approximate TSP optimizer to filter out promising samples before solving them optimally. Tests show that both TSP optimizers (Concorde and OR-tools) perform relatively similar in terms of CPU-time on the given instance set: Its TSP's are often quite short (5 – 20 locations) and there is a significant amount of static CPU-time software overhead relative to the actual TSP optimization for these cases. The main optimization result is that the SMD heuristic proves useful, at least in terms of computational efficiency, the way it is used within the SBI optimizer and for the OBP version at hand.

6. Conclusion

We investigated computational efficiency in approximate Order Batching Problem (OBP) optimization. In previous work, computational efficiency has not been given enough attention, especially when considering unconventional warehouse layouts and vehicle types (Aerts et al., 2021). It is an important topic that affects operational costs both directly and indirectly, however. In experiments we studied the computational efficiency of two approximate optimizers, Single Batch Iterated (SBI) and Metropolis Batch Sampling (MBS). They both begin by obtaining an initial solution using the Sequential Minimal Distance (SMD) heuristic. SBI then improves on this solution by rerunning the SMD selection procedure using a semi-stochastic seed-order index, whereas MBS improves on the initial solution using a Metropolis algorithm.

For OBP instances with up to 100 orders and a few seconds of CPU-time, both optimizers yield distances only a few percentage points higher than results obtained at timeout (or optimal results where such are available). The result corroborates previous research claims: Fast approximate optimization is a practicable choice in common OBP scenarios (Bozer & Kile, 2008; Kulak et al., 2012).

For larger instances, with 100 – 1000 orders, more time is required to obtain similar optimization gains. The standardized absolute distance saved through the optimization procedure grows similarly for all instance sizes. In SBI's case this can be explained since weak batches (with products located far from each other) are only constructed whenever there are few orders left to select from (SMD prevents this in other cases). This phenomenon occurs an equal number of times regardless of instance size and the amount of possible solution improvement in larger instances is thus relatively low. MBS does not face this particular issue, but on the other hand it has no mechanism to reduce the vast search space. MBS generally performs worse than SBI within the 5 minute timeout, particularly on larger instances.

Regardless of instance size, we conclude that that the value in spending significantly more CPU-time to obtain a result a few percentage points better than a baseline, must be weighed against the less measurable and indirect costs that come with lower computational efficiency. Although several authors have problematized large CPU-time requirements for OBP optimization (Bozer & Kile, 2008; Kulak et al., 2012; Valle & Beasley, 2019), it is challenging to

judge optimization efficiency generally due to the large variability of OBP usecases (Section 2).

For future work we believe the investigation can be widened to include more optimizers. MBS could be replaced by similar but more biased MCMC algorithms, such as Simulated Annealing (Rajasekaran & Reif, 1992) or Basin Hopping (Wales & Doye, 1997). Heuristics to add even more bias to these algorithms might be needed, however. Examples include mode-jumping (Tak et al., 2018) and restarts (Yu et al., 2021), which prevent convergence on local minima. Also, the number of required samples (N) for convergence could be estimated for various MCMC algorithms by calculating covariance between generated samples. That way, maximum CPU-time can be set in a more informed manner. For SBI, the Sequential Minimal Distance (SMD) heuristic could be replaced by alternatives which may be more suitable for the unconventional layout. We believe there are significant savings to be made in optimization if more memory is allocated to store and reuse parts of expensive computations. Modeling of OBP's and data-driven performance evaluation are also of primary importance. Currently there exists no standard format for OBP benchmark datasets and this poses a serious threat to scientific reproducibility. Since there are many possible versions of OBP's, the community needs to discuss how a standard format for OBP benchmark data can be designed to balance realism with simplicity and reproducibility.

Compliance with ethical standards: Funding: This work was partially supported by the Wallenberg AI, Autonomous Systems and Software Program (WASP). Conflict of Interest: The authors declare that they have no conflict of interest. This article does not contain any studies with human participants or animals performed by any of the authors.

References

Aerts, B., Cornelissens, T., & Sörensen, K. (2021). The joint order batching and picker routing problem: Modelled and solved as a clustered vehicle routing problem. *Computers & Operations Research*, *129*, 105168. https://doi.org/10.1016/j.cor.2020.105168

Applegate, D., Cook, W., Dash, S., & Rohe, A. (2002). Solution of a Min-Max Vehicle Routing Problem. *INFORMS Journal on Computing*, *14*, 132–143.

Applegate, D. L., Bixby, R. E., Chvatal, V., & Cook, W. J. (2006). *The traveling salesman problem: A computational study*. Princeton university press.

Azadnia, A. H., Taheri, S., Ghadimi, P., Samanm, M. Z. M., & Wong, K. Y. (2013). *Order Batching in Warehouses by Minimizing Total Tardiness: A Hybrid Approach of Weighted Association Rule Mining and Genetic Algorithms*. *Scientific World Journal*.

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

Briant, O., Cambazard, H., Cattaruzza, D., Catusse, N., Ladier, A.-L., & Ogier, M. (2020). An efficient and general approach for the joint order batching and picker routing problem. *European Journal of Operational Research*, *285*(2), 497–512. https://doi.org/10.1016/j.ejor.2020.01.059

Bué, M., Cattaruzza, D., Ogier, M., & Semet, F. (2019). *A Two-Phase Approach for an Integrated Order Batching and Picker Routing Problem* (pp. 3–18).

Cergibozan, Ç., & Tasan, A. (2020). Genetic algorithm based approaches to solve the order batching problem and a case study in a distribution center. *Journal of Intelligent Manufacturing*, 1–13. https://doi.org/10.1007/s10845-020-01653-3

Chen, M.-C., & Wu, H.-P. (2005). An association-based clustering approach to order batching considering customer demand patterns. *Omega*, *33*(4), 333–343. https://doi.org/10.1016/j.omega.2004.05.003

Cordeau, J.-F., Laporte, G., Savelsbergh, M., & Vigo, D. (2007). Vehicle Routing. In *Transportation, handbooks in operations research and management science* (Vol. 14, pp. 195–224).

Defryn, C., & Sörensen, K. (2017). A fast two-level variable neighborhood search for the clustered vehicle routing problem. *Computers & Operations Research*, *83*, 78–94. https://doi.org/10.1016/j.cor.2017.02.007

Esposito, C., Castiglione, A., & Choo, K.-K. R. (2016). Challenges in Delivering Software in the Cloud as Microservices. *IEEE Cloud Computing*, *3*(5), 10–14. https://doi.org/10.1109/MCC.2016.105

Gademann, A. J. R. M. (noud), Van Den Berg, J. P., & Van Der Hoff, H. H. (2001). An order batching algorithm for wave picking in a parallel-aisle warehouse. *IIE Transactions*, *33*(5), 385–398.

Henn, S. (2012). Algorithms for on-line order batching in an order picking warehouse. *Computers & Operations Research*, *39*(11), 2549–2563.

Henn, S., Koch, S., Doerner, K. F., Strauss, C., & Wäscher, G. (2010). Metaheuristics for the order batching problem in manual order picking systems. *Business Research*, *3*(1), 82–105.

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

Ho, Y.-C., Su, T.-S., & Shi, Z.-B. (2008). Order-batching methods for an order-picking warehouse with two cross aisles. *Computers & Industrial Engineering*, *55*(2), 321–347.

Jiang, X., Zhou, Y., Zhang, Y., Sun, L., & Hu, X. (2018). Order batching and sequencing problem under the pick-and-sort strategy in online supermarkets. *Procedia Computer Science*, *126*, 1985–1993.

Kruk, S. (2018). *Practical Python AI Projects: Mathematical Models of Optimization Problems with Google OR-Tools*. Apress.

Kulak, O., Sahin, Y., & Taner, M. E. (2012). Joint order batching and picker routing in single and multiple-cross-aisle warehouses using cluster-based tabu search algorithms. *Flexible Services and Manufacturing Journal*, *24*(1), 52–80. https://doi.org/10.1007/s10696-011-9101-8

Li, J., Huang, R., & Dai, J. B. (2017). Joint optimisation of order batching and picker routing in the online retailer's warehouse in China. *International Journal of Production Research*, *55*(2), 447–461. https://doi.org/10.1080/00207543.2016.1187313

Mackay, D. J. C. (1998). Introduction to Monte Carlo Methods. *Learning in Graphical Models*.

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

Naumenko, T., & Petrenko, A. (2021). Analysis of Problems of Storage and Processing of Data in Serverless Technologies. *Technology Audit and Production Reserves*, *2*(2), 58.

Oxenstierna, J., Malec, J., & Krueger, V. (2021). Layout-Agnostic Order-Batching Optimization. *International Conference on Computational Logistics*, 115–129.

Oxenstierna, J., Malec, J., & Krueger, V. (2022). Analysis of Computational Efficiency in Iterative Order Batching Optimization. *Proceedings of the 11th International Conference on Operations Research and Enterprise Systems - ICORES*, 345–353. https://doi.org/10.5220/0010837700003117

Rajasekaran, S., & Reif, J. H. (1992). Nested annealing: A provable improvement to simulated annealing. *Theoretical Computer Science*, *99*(1), 157–176. https://doi.org/10.1016/0304-3975(92)90177-H

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

Rensburg, L. J. van. (2019). *Artificial intelligence for warehouse picking optimization—An NP-hard problem* [Master's Thesis]. Uppsala University.

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

Scholz, A., Schubert, D., & Wäscher, G. (2017). Order picking with multiple pickers and due dates – Simultaneous solution of Order Batching, Batch Assignment and Sequencing, and Picker Routing Problems. *European Journal of Operational Research*, *263*(2), 461–478. https://doi.org/10.1016/j.ejor.2017.04.038

Sharp, G. P., & Gibson, D. R. (1992). Order batching procedures. *European Journal of Operational Research*, *58*.

Tak, H., Meng, X.-L., & Dyk, D. A. van. (2018). A Repelling–Attracting Metropolis Algorithm for Multimodality. *Journal of Computational and Graphical Statistics*, *27*(3), 479–490. https://doi.org/10.1080/10618600.2017.1415911

Valle, C. A., & Beasley, B. A. (2019). *Order batching using an approximation for the distance travelled by pickers*. *European Journal of Operational Research*.

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

Wales, D. J., & Doye, J. P. K. (1997). Global Optimization by Basin-Hopping and the Lowest Energy Structures of Lennard-Jones Clusters Containing up to 110 Atoms. *Journal of Physical Chemistry A*, *101*, 5111–5116.

Yu, V. F., Winarno, Maulidin, A., Redi, A. A. N. P., Lin, S.-W., & Yang, C.-L. (2021). Simulated Annealing with Restart Strategy for the Path Cover Problem with Time Windows. *Mathematics*, *9*(14). https://doi.org/10.3390/math9141625

Appendix

Table 1: Aggregation of SBI test-instance results into categories based on number of orders in the OBP's. Within each category the average over all results is shown. Whenever the specified time is not relevant or the optimizer failed to obtain a result within its scope, a minus sign (-) is shown. The distances shown are standardized.

# Orders	# Products	# Batches in solution	Baseline distance	Time to obtain baseline	Improvement percentage, after					
					5 s	10 s	30 s	60 s	120 s	300 s
4-5	15.93	1.09	123.25	0.03	2.99	3.01	-	-	-	-
6-7	27.56	1.80	143.12	0.18	3.23	4.36	-	-	-	-
8-9	39.43	3.78	153.77	0.21	3.47	5.28	-	-	-	-
10-11	71.90	4.16	192.00	0.37	3.71	3.92	-	-	-	-
12-15	75.67	4.90	191.47	0.33	3.95	5.62	-	-	-	-
16-20	110.15	5.58	232.89	0.35	4.19	4.57	-	-	-	-
21-25	145.50	5.29	266.29	0.33	4.42	4.42	-	-	-	-
26-30	161.79	6.52	284.31	0.49	4.66	4.95	-	-	-	-
31-40	221.97	6.60	348.57	0.48	4.90	4.90	-	-	-	-
41-50	272.44	7.24	429.48	0.43	5.14	5.74	-	-	-	-
51-60	306.41	8.10	471.01	0.56	5.38	5.38	-	-	-	-
61-70	331.36	8.94	509.58	0.57	5.62	6.50	-	-	-	-
71-80	360.14	8.80	544.29	0.65	5.86	6.21	-	-	-	-
81-90	390.52	9.65	592.94	0.77	6.10	8.01	-	-	-	-
91-100	439.70	9.46	667.07	0.68	3.03	3.62	3.90	-	-	-
101-110	379.96	10.74	632.89	0.54	2.84	3.40	3.83	-	-	-
111-120	396.37	12.58	745.79	0.79	2.65	3.21	3.74	3.75	-	-
121-130	418.97	11.15	859.19	1.22	2.43	3.03	3.62	3.66	-	-
131-140	437.75	13.41	972.50	1.36	2.23	2.81	3.53	3.61	3.92	-
141-150	463.67	13.22	1091.84	1.11	2.02	2.63	3.42	3.49	3.94	-
151-160	473.42	14.57	1200.46	1.51	1.82	2.43	3.33	3.45	4.00	-
161-170	494.31	13.62	1305.03	1.77	1.64	2.46	3.24	3.37	4.07	5.24
171-180	517.93	14.35	1425.75	1.93	1.42	2.55	3.11	3.32	4.12	5.41
181-190	539.55	13.75	1543.44	1.91	1.21	1.83	3.01	3.23	4.16	5.81
191-200	555.12	14.41	1649.98	2.06	1.01	1.28	2.94	3.14	4.20	5.71
201-250	662.06	16.16	1853.66	2.10	0.81	1.43	1.53	3.11	4.22	5.54
251-300	802.01	19.34	2087.13	2.22	0.61	1.46	1.51	2.69	3.92	4.92
301-350	989.22	22.95	2369.19	2.41	0.60	1.31	1.52	2.13	3.03	4.52
351-400	1326.53	24.11	2805.68	3.16	0.74	1.15	1.38	1.56	1.75	4.95
401-450	1438.57	26.00	3006.29	2.69	0.53	0.53	0.77	1.01	1.25	3.46
451-500	1581.54	27.48	3249.08	3.30	0.20	0.50	0.58	0.60	1.65	2.73
501-550	1694.52	27.80	3453.46	4.07	0.24	1.02	1.23	1.45	2.70	3.89
551-600	1989.83	29.74	3843.07	4.38	0.30	1.58	1.79	2.03	2.27	2.50
601-650	2149.81	35.85	4093.18	5.32	-	1.06	1.56	2.27	2.60	3.14
651-700	2316.35	39.98	4348.14	5.02	-	1.32	1.36	1.38	1.42	1.43
701-750	2660.46	46.27	4796.14	5.92	-	0.89	1.01	1.12	1.33	1.59
751-800	3019.32	50.12	5248.71	6.05	-	0.45	0.98	1.61	2.12	2.64
801-850	3154.55	55.04	5472.26	6.26	-	0.61	0.63	0.65	1.74	1.75
851-900	3290.79	59.79	5708.32	7.30	-	1.04	1.08	1.10	1.11	1.87
901-950	3497.78	64.09	6199.37	7.80	-	0.29	0.47	0.67	1.60	1.97
951-1000	3760.56	71.14	6568.42	8.30	-	0.80	0.96	1.46	1.78	2.54

Table 2: Aggregation of MBS test-instance results into categories based on number of orders in the OBP's. Within each category the average over all results is shown. Whenever the specified time is not relevant or the optimizer failed to obtain a result within its scope. The distances shown are standardized.

# Orders	# Products	# Batches in solution	Baseline distance	Time to obtain baseline	Improvement percentage, after					
					5 s	10 s	30 s	60 s	120 s	300 s
4-5	15.93	1.09	123.25	0.02	3.00	3.54	-	-	-	-
6-7	27.56	1.80	143.12	0.19	2.95	3.04	-	-	-	-
8-9	39.43	3.69	155.02	0.19	3.32	3.45	-	-	-	-
10-11	71.90	4.06	188.36	0.41	3.39	3.70	-	-	-	-
12-15	75.67	4.98	196.83	0.34	3.01	3.13	-	-	-	-
16-20	110.15	5.58	233.62	0.37	2.23	2.87	-	-	-	-
21-25	145.50	5.87	275.69	0.39	3.23	3.63	-	-	-	-
26-30	161.79	6.52	284.31	0.39	3.95	3.93	-	-	-	-
31-40	221.97	6.60	348.57	0.37	2.47	3.16	-	-	-	-
41-50	272.44	7.24	429.48	0.36	3.06	3.14	-	-	-	-
51-60	306.41	8.70	491.63	0.47	2.77	3.66	-	-	-	-
61-70	331.36	8.94	509.58	0.55	2.58	3.02	-	-	-	-
71-80	360.14	8.83	531.02	0.40	1.95	2.40	2.53	-	-	-
81-90	390.52	9.60	582.97	0.95	2.56	2.95	2.98	-	-	-
91-100	439.70	9.46	667.07	0.76	2.28	2.39	3.69	-	-	-
101-110	379.96	11.76	644.22	0.60	1.88	2.16	2.40	2.59	-	-
111-120	396.37	12.50	748.19	0.89	1.46	2.29	2.60	2.75	-	-
121-130	418.97	11.25	854.23	1.41	2.14	2.19	2.53	2.88	2.91	-
131-140	437.75	13.46	971.66	1.38	1.69	1.77	1.97	2.06	2.13	-
141-150	463.67	13.24	1109.84	1.02	1.25	1.37	1.84	2.05	2.22	-
151-160	473.42	14.57	1200.46	1.23	1.58	1.89	1.91	2.15	2.18	-
161-170	494.31	13.57	1334.74	1.59	1.40	1.92	2.02	2.05	2.14	2.65
171-180	517.93	14.35	1425.75	1.98	1.23	2.31	2.15	2.50	2.58	2.68
181-190	539.55	13.77	1552.33	2.06	1.18	1.60	1.72	2.21	2.28	2.41
191-200	555.12	14.51	1660.19	2.00	0.95	1.30	1.50	1.59	1.85	2.05
201-250	662.06	16.16	1853.66	2.35	0.63	1.02	1.07	1.66	1.91	2.32
251-300	802.01	19.99	2192.60	2.18	0.34	0.84	1.46	1.57	1.64	2.00
301-350	989.22	23.38	2478.75	2.37	0.55	0.71	1.20	1.27	1.28	1.72
351-400	1326.53	24.01	2811.58	3.01	0.56	0.69	1.23	1.29	1.35	1.78
401-450	1438.57	26.00	3006.29	2.67	0.13	0.13	0.27	0.27	1.34	1.58
451-500	1581.54	27.48	3249.08	3.22	0.10	0.16	0.22	0.27	0.76	1.56
501-550	1694.52	27.80	3453.46	4.35	0.07	0.08	0.37	0.49	1.59	2.00
551-600	1989.83	28.29	3743.11	4.73	0.10	0.53	1.09	1.15	1.21	1.76
601-650	2149.81	35.85	4093.18	5.51	0.34	0.59	0.65	0.76	0.83	1.15
651-700	2316.35	40.05	4431.19	5.28	-	0.40	0.61	0.67	1.08	1.31
701-750	2660.46	48.19	4876.53	6.15	-	0.25	0.58	0.59	0.86	1.09
751-800	3019.32	50.29	5260.92	6.12	-	0.12	0.40	0.49	0.92	1.10
801-850	3154.55	55.00	5468.35	5.95	-	0.38	0.39	0.42	0.49	0.69
851-900	3290.79	59.75	5620.73	6.96	-	0.26	0.28	0.30	0.35	0.83
901-950	3497.78	66.15	6214.21	7.55	-	0.15	0.24	0.28	0.64	1.09
951-1000	3760.56	70.11	6551.99	8.88	-	0.29	0.42	0.47	0.90	1.37

Storage Assignment using Nested Metropolis Sampling and Approximations of Order Batching Travel Costs

Johan Oxenstierna, Jacek Malec and Volker Krueger

Abstract

The Storage Location Assignment Problem (SLAP) is of central importance in warehouse operations. An important research challenge lies in generalizing the SLAP such that it is not tied to certain order-picking methodologies, constraints, or warehouse layouts. We propose the OBP-based SLAP, where the quality of a location assignment is obtained by optimizing an Order Batching Problem (OBP). For the optimization of the OBP-based SLAP, we propose a nested Metropolis algorithm. The algorithm includes an OBP-optimizer to obtain the cost of an assignment, as well as a filter which approximates OBP costs using a model based on the Quadratic Assignment Problem (QAP). In experiments, we tune two key parameters in the QAP model, and test whether its predictive quality warrants its use within the SLAP optimizer. Results show that the QAP model's per-sample accuracy is only marginally better than a random baseline, but that it delivers predictions much faster than the OBP optimizer, implying that it can be used as an effective filter. We then run the SLAP optimizer with and without using the QAP model on industrial data. We observe a cost improvement of around 23% over 1 hour with the QAP model, and 17% without it. We share results for public instances on the TSPLIB format.

1. Introduction

Charris et al. (2018) gives the following definition of a Storage Location Assignment Problem (SLAP): The "allocation of products into a storage space and optimization of the material handling (…) or storage space utilization [costs]". The relationship between material handling costs, on the one hand, and storage assignment, on the other, can be showcased in an example: If a vehicle needs to pick a set of products, its travel cost clearly depends on where the products are stored in the warehouse. At the same time, the development of an effective storage strategy needs to consider various features in material handling, such as vehicle constraints, traffic conventions and picking methodologies.

In this paper, we work with a version of the SLAP which is particularly generalizable. Kübler et al. (2020), name this version the "joint storage location assignment, order batching and picker routing problem". The main characteristic of this problem is the inclusion of two optimization problems in the SLAP:

1. The *Order Batching Problem* (OBP), where vehicles are assigned to carry sets of *orders* (an order is a set of products) (Koster et al., 2007).
2. The *Picker Routing Problem*, where a short picking path of a vehicle is found for the products that the vehicle is assigned to pick. The Picker Routing Problem is a Traveling Salesman Problem (TSP) applied in a warehouse environment (Ratliff & Rosenthal, 1983).

Henceforth, we refer to this version as the OBP-based SLAP. A key advantage of using the OBP within the SLAP is the added flexibility and generality of the *order* on a conceptual level: For example, optimizing the OBP-based SLAP gives opportunity to also optimize the TSP-based SLAP (Oxenstierna et al., 2023). When it comes to product locations, the sole difference between the OBP and the OBP-based SLAP is that locations for all products are assumed fixed in the former while, in the latter, they are assumed mutable (for a subset of locations in our case).

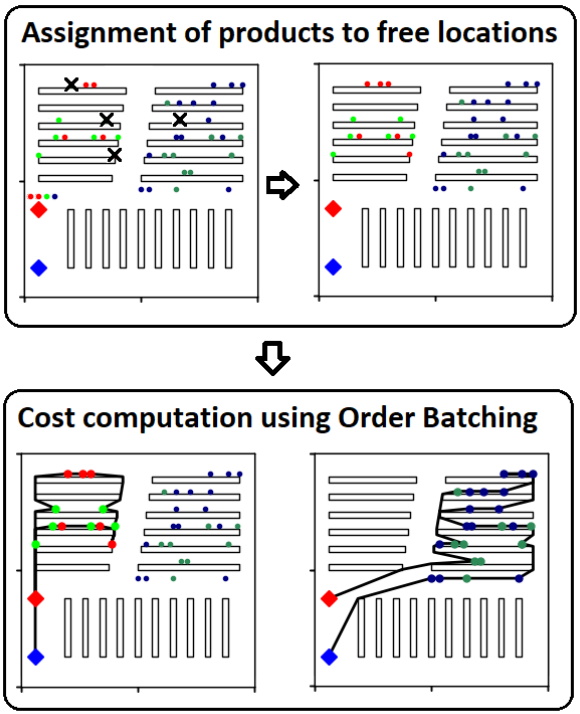

Figure 1: Example storage assignment of four products and subsequent order-picking for the SLAP model used in the paper. Rectangles denote warehouse racks. Red and blue diamonds denote origin/destination for picking paths. Colored dots denote products and the four orders they belong to. Black crosses denote available locations for the new products. Note that products are often more spread out than what is shown in this example.

It is of scientific importance to be able to compare optimization approaches and solutions. For the SLAP, this is made difficult by the many versions of the problem. As the extensive literature review by Charris et al. (2018) shows, there is little consensus regarding which versions are more important, or specifically, which features would represent a standardized version. Examples of such features are dynamicity, warehouse layout, vehicle types, cost functions, reassignment scenarios and picking methodologies. There is also a shortage of benchmark datasets for any version of the SLAP, which prevents the reproducibility of experiments (Aerts et al., 2021; Kofler et al., 2014). As part of our contribution for a standardized version, we suggest a modified TSPLIB format (Reinelt, 1991) (Section 6). There are several ways in which to balance between simplicity, reproducibility and industrial applicability when developing SLAP versions and corresponding instances, however. From

the generalization perspective, our model is advantageous in two main areas: Order-picking methodology and warehouse layout. But it is weak in two other areas: dynamicity and reassignment scenarios. We describe the meaning of these choices further in the light of prior work (Section 2) and in our problem formulation (Section 4). We invite the community to debate which features are more or less important for a standardized version.

In Section 5, we introduce our SLAP optimizer. It is based on the Metropolis algorithm, a type of Markov Chain Monte Carlo (MCMC) method. A core feature of the optimizer is that the quality of a location assignment candidate is retrieved by optimizing an OBP. Due to the OBP's NP-hardness, it must be optimized in a way that trades off solution quality with CPU-time. For this purpose, we use an OBP optimizer with a high degree of computational efficiency (Oxenstierna et al., 2022). Within the SLAP optimizer, the OBP optimizer is still computationally expensive, and we show that it can be assisted by fast cost approximations from a Quadratic Assignment Problem (QAP) model. Finally, we test the performance of the SLAP optimizer with and without inclusion of the QAP approximations. Cost improvements are around 23% over 1 hour with the QAP model, and 17% without. In summary, we make three concrete contributions:

1. *Formulation of an OBP-based SLAP optimization model and a corresponding benchmark instance standard.*
2. *QAP approximation model to predict OBP travel costs and experiments on generated instances to test whether the use of QAP approximations within a SLAP optimizer can be justified.*
3. *An OBP-based SLAP optimizer (QAP-OBP) and experiments on industry instances to test its computational efficiency. Comparison of results with and without usage of QAP approximations.*

2. Related work

This section goes through general strategies for conducting storage location assignment, as well as ways in which their quality can be evaluated. Various SLAP formulations and proposed optimization algorithms are covered. Our primary focus will be on the standard picker-to-parts arrangement. We specifically refer to the work of Kübler et al. (2020), as their proposed model aligns with ours.

There exist numerous general strategies for conducting storage location assignment (Charris et al., 2018). Three key strategies are Dedicated, Class-based and Random:

- *Dedicated*: Each product is assigned to a specific location which never changes. This strategy is suitable if the product collection changes rarely and simplicity is desired. Additionally, human pickers can leverage this strategy by familiarizing themselves with specific products and their corresponding locations, which might speed up their picking (Zhang et al., 2019).
- *Random*: Each product can be assigned any available location in the warehouse. This is suitable whenever the product collection changes frequently.
- *Class-based (zoning)*: The warehouse is partitioned into sections, and the products are classified based on their demand. Each class is assigned a zone. The outline of the zone can be regarded as dedicated in that it does not change, whereas the placement of each product in a zone is assumed to be random (Mantel et al., 2007). Class-based storage assignment can therefore be regarded as a middle ground between dedicated and random.

The quality of a location assignment is commonly evaluated based on some model of aggregate travel cost. For this purpose, a simplified simulation of order-picking in the warehouse can be used (Charris et al., 2018; Mantel et al., 2007). Some proposals include the simulation of order-picking by the Cube per Order Index (COI) (Kallina & Lynn, 1976). COI includes the volume of a product and the frequency with which it is picked (historically or future-forecasted). Products with high pick frequency and relatively low volume are subsequently assigned to locations close to the depot. Since orders may contain products which are not located close to each other, COI is only adequate for order-picking scenarios where orders contain one product and vehicles carry

one product at a time. This may be sufficient for pallet picking or when certain types of robots are used (Azadeh et al., 2019). Mantel et al. (2007), introduced Order Oriented Slotting (OOS) where the number of products in an order may be greater than one. A similar model to OOS is used by Fontana & Nepomuceno (2017), Lee et al. (2020) and Žulj et al. (2018). The picking cost of an order in OOS can in some cases be modeled using a Quadratic Assignment Problem (QAP) (Mantel et al., 2007). The QAP computes the sum of element-wise products of weights and frequencies (Abdel-Basset et al., 2018) and for an order this can be translated into distances between products and how often they are picked. Nevertheless, a QAP on its own is often not sufficient to model a SLAP without extensive use of heuristics and constraints for warehouse layouts and picking methodologies (Mantel et al., 2007). For a layout-agnostic OBP-based SLAP, graph-based QAP techniques could be attempted, but hitherto they have only been applied on related problems (X. Wu et al., 2021; Zhou & De la Torre, 2016).

There is only limited research on SLAPs where vehicles are expected to carry multiple orders and where an Order Batching Problem (OBP) is integrated into the SLAP optimization process. One example is Xiang et al. (2018) and Yang, (2022), who use this approach in a robotic warehouse where the vehicles are pods or mobile racks, which is not easily comparable to a picker-to-parts system. Another example is Kübler et al. (2020), which we look closer at below.

Travel distance or time are commonly used to evaluate SLAP solution quality in the above mentioned models, but there are several alternatives and extensions. Lee et al. (2020), for example, study the effect of location assignment and traffic congestion in a warehouse. Assigning too many products to locations close to the depot (the goal in common COI) may lead to traffic congestion, which should ideally be considered in an industrial model. Lee et al. (2020), formulate Correlated and Traffic Balanced Storage Assignment (C&TBSA) as a multi-objective problem with travel cost on the one hand, and traffic congestion avoidance on the other. Larco et al. (2017), include worker welfare in their evaluation of solution quality. If picking is conducted by humans who move products from shelves onto a vehicle, the weight and volume, as well as the height of the shelf the product is placed on, can have an impact on worker welfare. Parameters such as "ergonomic loading," "human energy expenditure," or "worker discomfort" (Charris et al., 2018) can be used to quantify worker welfare.

The SLAP can be categorized into two main groups based on the number of location assignments required. Either the assignment is a “re-warehousing”

operation, which means that a large portion of the warehouse's products are (re)assigned (Kofler et al., 2014). Often, however, only a small subset of products are (re)assigned, and this is referred to as "healing" (Kofler et al., 2014). Solution proposals involving healing often look closely at different types of scenarios for carrying out initial assignments for new products in the warehouse, or reassignments for products already in the warehouse. Kübler et al. (2020), propose four such scenarios.

I. *Empty storage location*: A product is assigned to a previously unoccupied location.

II. *Direct exchange*: A product changes location with another product.

III. *Indirect exchange 1*: A product is moved to another location which is occupied by another product. The latter product is moved to a third, empty location.

IV. *Indirect exchange 2*: A product is moved to a new location which is occupied by a second product. The second product is moved to a new location which is occupied by a third product. The third product is moved to the original location of the first product.

The above scenarios are all associated with varying levels of effort, ranging from the lightest in scenario I, to the heaviest in IV. Kübler et al. quantify these efforts by including both physical and administrative times, which are transformed to effort terms by proposed proportionalities.

Concerning SLAP optimizers, proposals include models capable of obtaining optimal solutions, such as Mixed Integer Linear Programming (MILP), dynamic programming and branch and bound algorithms (Charris et al., 2018). The warehouse environment is often simplified to a significant degree when optimal solutions are sought (Charris et al., 2018; Garfinkel, 2005; Kofler et al., 2014; Larco et al., 2017). The main simplification relates to order-picking using COI or OOS. Other simplifications involve limiting the number of products (Garfinkel, 2005), number of locations (J. Wu et al., 2014), or by requiring the conventional warehouse rack layout (Kübler et al., 2020). The conventional layout assumes Manhattan style blocks of aisles and cross-aisles, and it is used almost exclusively in existing literature on the SLAP (we are only aware of two exception cases using the "fishbone" and "cascade" layouts (Cardona et al., 2012; Charris et al., 2018).

Most proposed SLAP optimizers provide non-exact solutions using heuristics or meta-heuristics. One example is multi-phase optimization where the first phase proposes possible locations for products, and the second phase carries

out the assignments and evaluates them (Wutthisirisart et al., 2015). In Kübler et al. (2020), a heuristic zoning optimizer is used to generate location assignments, and a Discrete Evolutionary Particle Swarm Optimizer (DEPSO) is used to optimize an OBP for the evaluation of the assignments. DEPSO is a modification of a standard PSO algorithm that addresses the risk of convergence on local minima and allows for a discrete search space. Other heuristic or meta-heuristic approaches include Genetic and Evolutionary Algorithms (Ene & Öztürk, 2011; Lee et al., 2020), Ant Colony Optimization (Yingde & Smith, 2012) and Simulated Annealing (Zhang et al., 2019). If TSP optimization is desired within a SLAP, *S-shape* or *Largest Gap* algorithms (Roodbergen & Koster, 2001) are often utilized. For TSP-optimization on unconventional layouts with a pre-computed distance matrix, *Google OR-tools* or *Concorde* have been proposed (Oxenstierna et al., 2022; Rensburg, 2019).

Evaluating the quality of results in prior work is challenging due to the variability of SLAP models. Below are a few examples where result quality is judged based on a percentage saving in travel distance or time: For conventional warehouse layouts, reassignment costs and dynamic picking patterns, Kofler et al. (2014), report best savings around 21%. Kubler et al. (2020), report best savings around 22% in a similar scenario. Zhang et al. (2019) report best savings around 18% on simulated data with thousands of product locations, but without reassignment costs. In a similar setting, for a few hundred products, Trindade et al. (2022) report best savings around 33%.

3. Nested Metropolis sampling

The proposed optimizer (Section 5) is based on a nested Metropolis algorithm first introduced by Christen & Fox (2005). The Metropolis algorithm is a type of Markov Chain Monte Carlo (MCMC) method, which first draws a sample x_{i+1} based on a desired feature distance (excluding costs) to a previous sample x_i. The distance is given by some probability distribution $q(x_{i+1}|x_i)$, and it is usually chosen such that the distance between x_{i+1} and x_i is low with a high probability (Mackay, 1998). The *accept* probability is then computed based on some function that takes the costs of the new and previous samples as input (van Ravenzwaaij et al., 2018). Common Metropolis sampling assumes that there is only one cost function, f^*, and since we wish to include an approximation of this cost, f, we use a modification (Christen & Fox, 2005). Nested Metropolis sampling is shown in flowchart form in Figure 2.

After a first sample x_i has been initialized (i), a new sample x_{i+1} is generated (ii) and its cost approximated $f(x_{i+1})$ (iii). If the approximation is deemed strong enough (probabilistically) relative to $f(x_i)$, the sample is *promoted* (iv) to the next step where its ground-truth cost $f^*(x_{i+1})$ is computed (v). The accept filter (vi) is only used for promoted samples.

For a cost minimization problem, the promote and accept probabilities can be computed based on the following equations (Christen & Fox, 2005):

$$\alpha(x_{i+1}|x_i) = min\big(1, f(x_i)/f(x_{i+1})\big) \quad (1)$$

$$\alpha^*(x_{i+1}|x_i) = min\big(1, f^*(x_i)/f^*(x_{i+1})\big) \quad (2)$$

where $\alpha(x_{i+1}|x_i)$ denotes the promote probability and $\alpha^*(x_{i+1}|x_i)$ the accept probability.

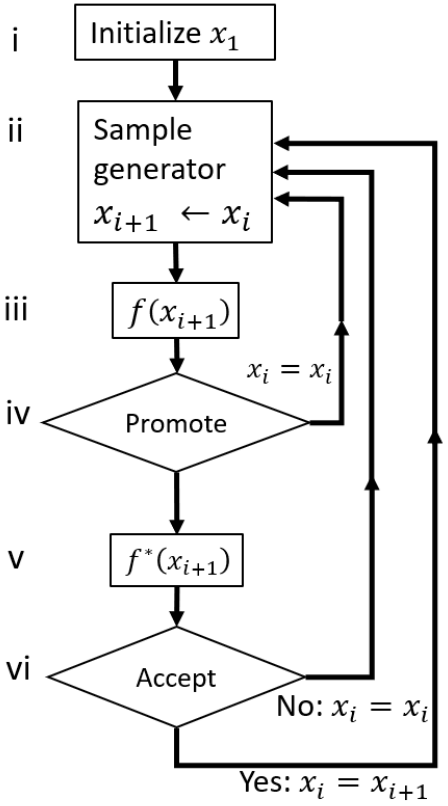

Figure 2: Nested Metropolis Sampling. The inner loop computes a cheap (in terms of CPU-time) approximation of a sample cost and if the approximation is strong, the sample is promoted to the outer loop where an expensive ground-truth cost is computed.

4. Problem formulation

4.1 Objective function

The objective function in the OBP-based SLAP is based on the ones formulated in Henn & Wäscher (2012) and Oxenstierna, van Rensburg, et al. (2021), i.e., the minimization of cost in an Order Batching Problem (OBP):

$$f^*(x) = min \sum_{b \in \mathcal{B}} D^x(b) a_{vb}, v \in V, \mathcal{B} \subset 2^{\mathcal{O}} \quad (3)$$

where $\mathcal{O}$ denotes orders, where $\mathcal{B}$ denotes batches and where $D^x(b)$ denotes the distance of a TSP solution, i.e., the distance needed to pick batch $b \in \mathcal{B}$. Batch b is a set of orders and $v \in V$ denotes a vehicle. Each vehicle can carry one batch and the number of orders that can fit in the batch is governed by vehicle capacity (such as dimensions, bins, number of orders or products). a_{vb} denotes a binary variable set to 1 if vehicle v is assigned to pick b and 0 otherwise. Orders consist of products $\mathcal{O} \in 2^{\mathcal{P}}$, where each product $p \in \mathcal{P}$ is a tuple consisting of a unique key (Stock Keeping Unit), a Cartesian location $loc(p)$, and a positive quantity of how many p are available at $loc(p)$. The locations of all products are given by location assignment vector x, where the elements represent products and the indices locations (each index is mapped to a Cartesian coordinate).

The mapping of location keys to coordinates and computation of distances between pairs of locations is based on a digitization pipeline for warehouses on any 2D obstacle layout and usage of the Floyd-Warshall graph algorithm. Details on this digitization pipeline and the OBP (including TSP-optimization for $D^x(b)$ and usage of vehicle capacity in a_{vb}) are beyond the scope of this paper, so for specifics we refer to Oxenstierna, van Rensburg, et al., (2021) and Rensburg (2019).

The difference between the OBP and the OBP-based SLAP mainly concerns product locations. In Oxenstierna, van Rensburg, et al. (2021) each product $p \in \mathcal{P}$ "has a [fixed] location", meaning that x in $f^*(x)$ is immutable. In the OBP-based SLAP, however, a subset of products $\mathcal{P}_s \subset \mathcal{P}$ do not have fixed locations, which means that some elements in x can change indices in the vector. The OBP-based SLAP objective consists of finding location assignment x, such that the OBP in Equation 3 is minimized:

$$\underset{x}{\operatorname{argmin}} \sum_{b \in \mathcal{B}} D^x(b) a_{vb}, v \in V, \mathcal{B} \subset 2^{\mathcal{O}} \tag{4}$$

This objective lacks reassignment costs and is therefore a version of the "empty storage location" scenario I in Kübler et al. (2020) (Section 2). Exclusion of reassignment costs is motivated for this scenario, since the initial location assignment of new products in a warehouse is not optional, but a requirement. The other of Kübler et al.'s scenarios are all reassignments. Contrary to the initial assignments that we work with, reassignments can produce an increased travel cost, as potential gains in order-picking must be weighed against reassignment costs.

Although reassignments should ideally be included in a complete SLAP model, a standardized SLAP needs to be a trade-off between simplicity and complexity. In the TSP-based SLAP (Oxenstierna et al., 2023) it is shown that the optimization of reassignments is NP-hard and not easily combined with order-picking optimization within a SLAP. The TSP-based SLAP includes reassignments, but uses the TSP instead of the OBP to optimize order-picking. The OBP-based SLAP excludes reassignments, but includes the OBP, a significantly more challenging problem than the TSP. As is often the case in literature on the SLAP, choice of optimization model depends on which features are considered more important for the usecase at hand.

4.2 Fast OBP Cost Approximation

One key difficulty with the OBP-based SLAP is that the OBP poses a highly intractable problem. Even for relatively small OBP instances, a significant amount of CPU-time is needed to obtain substantial cost improvements (Kübler et al., 2020; Oxenstierna et al., 2022). In the case of the OBP-based SLAP, this means that it would require a large amount of CPU-time to minimize cost for many assignment candidates x (Equation 4). To alleviate this problem, we propose to include an approximation of $f^*(x)$:

$$f(x) = \sum_{p_1 \in \mathcal{P}} \sum_{\substack{p_2 \in \mathcal{P} \\ p_1 \neq p_2}} \sum_{l_1 \in \mathcal{L}_\mathcal{P}} \sum_{\substack{l_2 \in \mathcal{L}_\mathcal{P} \\ l_1 \neq l_2}} w_{p_1 p_2} \, d^x_{l_1 l_2} \times a(p_1, l_1) a(p_2, l_2) \tag{5}$$

where w denotes weight, where $d^x_{l_1 l_2}$denotes distance between two locations $l_1, l_2$and $a(p, l)$ a function which returns 1 if product p is located at location l and 0 otherwise. $f(x)$ is the element-wise summation of weights times distances. The cell values in the weight matrix represent the number of times two products, p_1, p_2, appear in the same order $o \in \mathcal{O}$. The (shortest) distances between all pairs of product locations are assumed pre-computed and stored in memory. We refer to Equation 5 as the Quadratic Assignment Problem (QAP) model. Note that we never minimize it. For the $f(x)$ approximation to be of use, we proceed to discuss how its ability to predict $f^*(x)$ can be evaluated.

Assuming a dataset of finite samples with approximated and ground truth costs $\big(x, f(x), f^*(x)\big) \in X, |X| \in \mathbb{Z}^+, f(x), f^*(x) \in \mathbb{R}^+$, the predictive quality of $f(X)$ versus $f^*(X)$ is obtainable through softmax cross-entropy (Bruch et al., 2019; Cao et al., 2007):

$$\mathbb{P}\big(f(x_i)\big) = \frac{e^{f(x_i)}}{\sum_{j=1}^{|X|} e^{f(x_j)}} \tag{6}$$

$$\mathbb{P}\big(f^*(x_i)\big) = \frac{f^*(x_i)}{\sum_{j=1}^{|X|} f^*\big(x_j\big)} \tag{7}$$

$$L = -\frac{1}{|X|} \sum_{(x_i, f(x_i), f^*(x_i))} \mathbb{P}\big(f^*(x_i)\big) \, log \mathbb{P}\big(f(x_i)\big) \tag{8}$$

where $\mathbb{P}\big(f(x_i)\big)$ and $\mathbb{P}\big(f^*(x_i)\big)$ denote the probabilities of approximate and ground truth costs of sample x_i, respectively, where $\big(x_i, f(x_i), f^*(x_i)\big) \in X$. L is the *loss*, i.e., a distance heuristic between $f(X)$ and $f^*(X)$. This approach can be extended into Normalized Discounted Cumulative Gain (NDCG) (Bruch et al., 2019):

$$NDCG = \frac{DCG}{IDCG} \tag{9}$$

$$DCG = \sum_{i=1}^{|X|} \frac{rel(\pi_{f(X)}(i))}{log_2(\pi_{f(X)}(i) + 1)} \tag{10}$$

$$IDCG = \sum_{i=1}^{|X|} \frac{rel(\pi_{f^*(X)}(i))}{log_2(\pi_{f^*(X)}(i) + 1)} \tag{11}$$

$\pi_{f(X)}$ is a *ranking* (an ordering of samples X according to their costs $f(X)$) and $rel(\pi_{f(X)}(i))$ is the relevance at rank $\pi_{f(X)}(i)$. $IDCG$ denotes an *ideal* value, where$rel(\pi_{f^*(X)}(1)) > rel(\pi_{f^*(X)}(2)) > \cdots > rel(\pi_{f^*(X)}(|X|))$, i.e., the case when the relevance of a sample corresponds with how highly it is ranked. Bruch et al. (2019) argue that NDCG is a stronger choice than softmax cross-entropy whenever cost is non-binary, which is the case in $f^*(x)$ (Equation 3). In Figure 13 (Appendix) an example is shown where NDCG is computed from $|X|$ samples.

In summary, we can quantify the predictive quality of the QAP model by its ability to rank a list of samples X against a ground truth ranking by the OBP optimizer. Since the nested Metropolis algorithm in Section 3 only stores two samples at any iteration, we modify the algorithm to instead work with more samples (Section 5). We also want to avoid the computation of $f^*(X)$ in each iteration, so in the optimization algorithm we only compute $f^*(argmin_x f(X))$. In Section 7, we conduct an experiment to test the validity of using the NDCG-based $f^*(argmin_x f(X))$ in SLAP optimization. In Section 6 we also discuss choice of datatype for the relevance values.

5. Optimization Algorithm

5.1 Overview

The proposed optimization algorithm includes three main modules: 1. a sample (location assignment) generator. 2. a fast cost approximator based on a model of the Quadratic Assignment Problem (QAP). 3. an Order Batching Problem (OBP) optimizer. In this paper, we mainly focus on how QAP approximations can be effectively utilized within the nested Metropolis sampler described in Section 3. In Sections 5.2 and 5.3, we therefore describe two main modifications. The final version (QAP-OBP) is shown in flowchart form in Figure 3 and pseudocode in Algorithm 1.

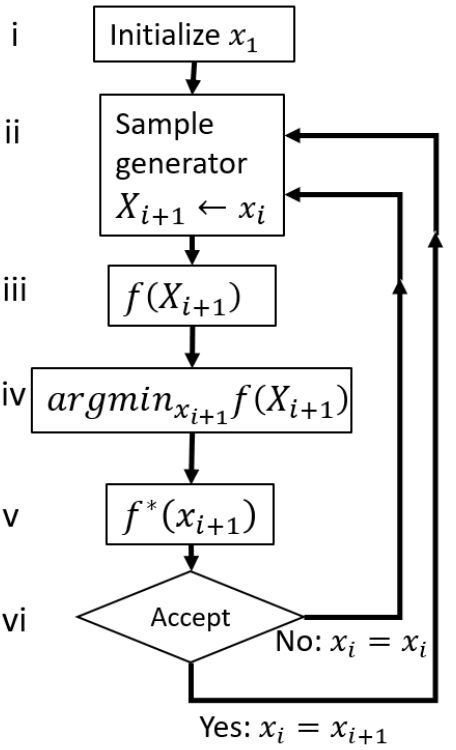

Figure 3: QAP-OBP optimization algorithm.

Algorithm 1 SLAP optimizer (QAP-OBP)

x, X: Sample(s) (location assignment(s)).
$f(X)$: Less accurate fast cost estimates (QAP).
$f^*(x)$: More accurate slow cost estimate (OBP).
N: Number of samples per iteration ($|X| = N$).
λ: Rate of change.
Δ: Cost-distance function.
α: Probability that sample x_{i+1} is *accepted*.
K: Number of iterations.
for $i = 1, ..., K$ **do**
    $X_{i+1} \leftarrow sample\_generator(N, \lambda, x)$
    $f(X_{i+1}) \leftarrow cost(X_{i+1})$
    $x_{i+1} \leftarrow argmin_{x_{i+1}} f(X_{i+1})$
    $f^*(x_{i+1}), f^*(x) \leftarrow cost^*(x_{i+1}, x)$
    $\alpha \leftarrow exp(-c_1 \Delta(f^*(x_{i+1}), f^*(x)))$
    $u \leftarrow \mathcal{U}(0, 1)$ // random uniform.
    if $u < \alpha$ **then** // sample *accepted*.
        $x \leftarrow x_{i+1}$
    end if
end for

Sample x contains both the assigned products (products already in the warehouse) and the unassigned products $\mathcal{P}_s$ (Section 4). x_1 is initialized such that products $\mathcal{P}_s$ are assigned free locations in x randomly without replacement. Choices for iterations K, the cost distance function Δ and constant c_1 are discussed in Section 7.

5.2 Sample Generator

The input to the sample generator (step ii in Figure 3) is a single sample x_i and the output is a list of new samples X_{i+1}. There are two main parameters in use by the sample generator. $N \in \mathbb{Z}^+$ dictates how many new samples are generated, i.e., $|X_{i+1}|$, and $\lambda \in \mathbb{R}^+$ dictates how much each new sample in X_{i+1}

differs from x_i. The way N and λ are utilized to generate new samples is shown in Algorithm 2.

Algorithm 2 Sample Generator

x: Sample (location assignment).
N: Number of new samples.
λ: Rate of change.
$X_{i+1} \leftarrow list()$: Empty list of new samples.
for $i = 1, ..., N$ **do**
    $x \leftarrow copy(x_i)$ // New sample.
    $m \leftarrow Pois(\lambda)$ // Number of changes.
    $\mathcal{P}_m \leftarrow remove\_m\_products(x, m)$
    for $p \in \mathcal{P}_m$ **do**
        $q \leftarrow p.get\_quantity()$
        $L \leftarrow possible\_locations(x, q)$
        $loc(p) \leftarrow assign(x, L)$
    end for
    $X_{i+1}.add(x)$
end for

Every time the sample generator is called, an empty list is first initialized. Then, for N iterations, a new sample x is generated by first copying x_i and then by computing m, the number of products for which the index in x can change. For m we use a truncated Poisson distribution with rate λ and upper bound $m \leq |\mathcal{P}_s|$. A uniform random selection of m products, $\mathcal{P}_m$, are then removed from x. For each $p \in \mathcal{P}_m$, a uniform random free index (either an empty location or an index holding a product in $\mathcal{P}_s$) in x is then selected such that the quantity (q) of the product does not exceed the location's capacity. After x has been filled, it is appended to X_{i+1}.

5.3 Promote and Accept Thresholds and Cost Computations

After a list of samples X_{i+1} has been generated (step ii in Figure 3), their costs are approximated using the QAP model (iii). The sample with the lowest cost approximation is then always promoted (iv). Steps ii, iii and iv in both the nested Metropolis sampler and QAP-OBP (Figure 2 and Figure 3, respectively) are the same considering that the final output is a single promoted sample. There are advantages and disadvantages of both versions regarding how they conduct this selection. In the nested Metropolis sampler in Figure 2, the promote probability depends on the ratio of approximated cost between previous and new single samples. In QAP-OBP, the sample generator is instead set to output $N = |X_{i+1}|$ candidates, followed by argmin (compare step iv in Figure 2 and 3). This modification simplifies evaluation of the QAP model's accuracy, since we can set up an experiment to compute OBP costs on the same samples (Figure 5). Generating multiple samples could also facilitate parallelization, which, for future work, could reduce the QAP model's CPU-time. The main consideration, however, is that it simplifies the original algorithm for a particularly complex optimization scenario, where it cannot be expected to behave according to Christen & Fox's (2005) performance guarantees. The problem with the original algorithm is that it assumes optimal $f^*(x)$ costs, but these are not generally available for OBPs (Oxenstierna et al., 2022) (as far as we are aware, there exists no proposal for how to obtain optimal results for but the smallest OBP instances within reasonable CPU-time). A relatively minor problem with the modification is that it requires tuning of the number of samples (N) that the sample generator is outputting each iteration. The reason we use a Metropolis algorithm instead of possibly more capable meta-heuristic alternatives, is mainly due to implementation. The Metropolis algorithm does not have many parameters which could be tuned based on iterations K (such as the temperature in Simulated Annealing) and therefore, a time-based condition can be used instead of K to terminate the algorithm (we will use this in Section 7.2.3).

Concerning computation of $f^*(x)$ we use the *Single Batch Iterated* (SBI) optimizer and its main features are its high computational efficiency and its ability to handle warehouses with unconventional rack layouts (Oxenstierna et al., 2022). OBP optimization and its internal use of TSP optimization, is beyond the scope of this paper, and we here treat SBI as a black-box which outputs a $f^*(x)$ for Equation 3. The sample x with the lowest $f^*(x)$ found is always stored throughout the optimization procedure (sample storage is omitted in Figure 2, Figure 3 and the pseudo-code).

6. Datasets

For this paper, we have generated and shared instances in L17_533[1], which are based on OBP instances in L6_203[2] and L09_251[3]. We also use data from a real warehouse (Aba Skol AB). The generated instances use the TSPLIB format (Reinelt, 1991) with certain amendments for the SLAP, including 6 types of warehouse obstacle layouts, various depot configurations, vehicle capacities and orders (see Figure 1 for an example of one of the layouts). L17_533 does not include any unidirectional travel rules, meaning that the distance between any two locations is equal both ways. The number of orders range between 4 to 1000 and number of products range between 10 to 3000. The products that are to be assigned a location, $\mathcal{P}_S$, are tagged as "SKUsToSlot" in the instance set. The "assignmentOptions" includes the available empty locations and how cost is to be computed (it is always set to the "empty storage location" scenario). For analysis, instances are categorized according to vehicle capacities, number of orders, products and parameters N and λ.

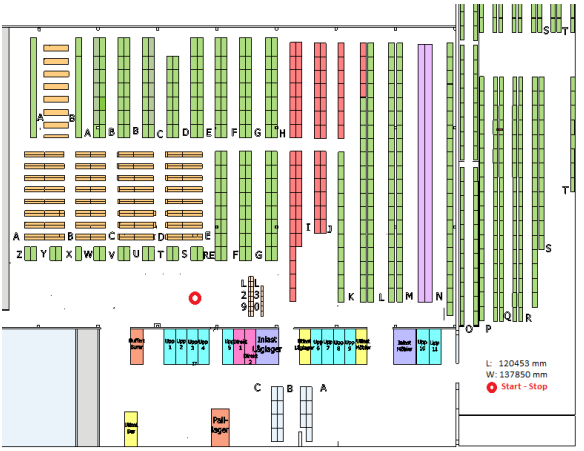

Figure 4: Top-view of the Aba Skol AB warehouse. The picking zones are color-coded. The red circle denotes the most commonly used depot location.

[1] https://github.com/johanoxenstierna/L17_533, collected 13-02-2023.

[2] https://github.com/johanoxenstierna/OBP_instances, collected 15-01-2023.

[3] https://github.com/johanoxenstierna/L09_251, collected 15-01-2023.

The industrial warehouse dataset (Figure 4) contains 210277 products in 37014 orders collected using batch picking over a 4-month period. There are 1289 pick-locations (in the graph representation) and most batches exist within one of six picking zones, but 24.4% include picks from several zones. As with the generated instances, shortest distances and paths between any two locations are assumed equal. For a proof of concept, we select product subsets from this data to be of relevance to warehouse management and real-world utility, on the one hand, and comparability to the generated instances, on the other. We build 150 subsets from 3-week periods with selections of between 50-1800 products for $\mathcal{P}$ and between 10 and 225 corresponding products for $\mathcal{P}_s$. The subset selection is random apart from that the products in a subset must exist within the same 3-week period. Number of free locations is given on a *per-product* basis, since each product has specific constraints regarding where it can be placed, and on average it varies between 50 – 481 locations. For parameters N and λ, we explore suitable values on the generated instances within shorter optimization runs, followed by longer runs with chosen constants on the real dataset.

7. Experiments

7.1 Overview and Constants

The experiments are divided into two parts. The first part involves tuning the QAP model and comparing its ability to rank SLAP assignment samples against an OBP ground truth model and a random baseline (Figure 5).

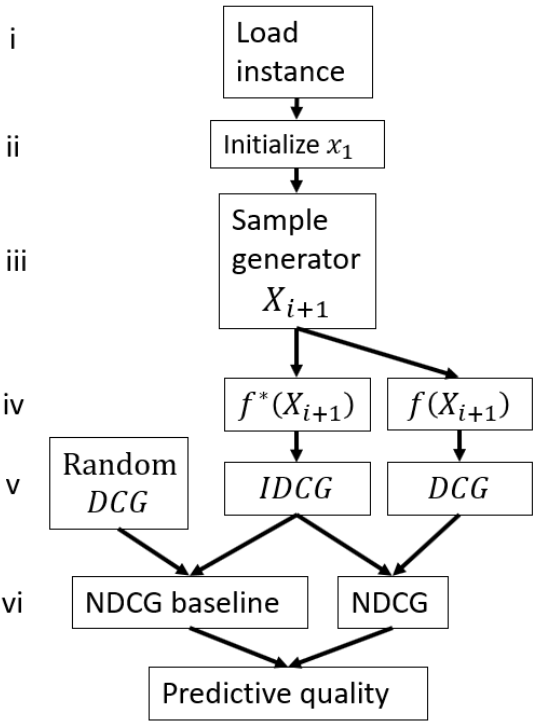

Figure 5: Steps involved to obtain QAP predictive quality on samples generated from an instance.

A SLAP test-instance (orders with products) is first loaded (i) and x_1 initialized (products $\mathcal{P}_s$ are assigned free locations in x_1 randomly) (ii). Then, N location assignments, X_{i+1}, are generated according to Algorithm 2 (iii). The cost of the generated assignments is estimated using the QAP model and the OBP optimizer SBI (iv). The samples and costs are used to compute IDCG and DCG (v). IDCG is computed from the ranking of costs according to the OBP

optimizer and DCG is computed from the ranking of costs according to the QAP model. A random DCG value is also pre-computed using the average of 10^6 random rankings. This random baseline represents the case when $f(X_{i+1})$ and $argmin_{x+1}\, f(X_{i+1})$ (steps iii and iv in Figure 3) cannot help produce a lower value in $f^*(x_{i+1})$ (step v) (Freund et al., 2003; Freund & Schapire, 1996). Relevance values $rel(\pi_{f^*(X)})$ and $rel(\pi_{f(X)})$ are chosen to be the ordinal ranks of samples x according to respective cost functions. For N samples, the values are $rel(\pi_{f^*(X)}) = (\pi_{f^*(X)}(N),\ \pi_{f^*(X)}(N-1), \dots,\ \pi_{f^*(X)}(1))$ and $rel(\pi_{f(X)}) = (\pi_{f(X)}(N),\ \pi_{f(X)}(N-1),\ \dots,\ \pi_{f(X)}(1))$ (this corresponds to the set up shown in Figure 13 in Appendix). The DCG value obtained from the QAP model is then used to compute NDCG according to Equation 9 (vi). The predictive quality is finally calculated by subtracting the achieved NDCG value with the random NDCG baseline, with a positive value implying that the QAP model is stronger. We also record the CPU-time needed for the QAP model and the OBP-optimizer, respectively. The tuning of the QAP model concerns parameters N (number of samples) and λ (rate of change for the samples) to maximize NDCG. We further investigate whether NDCG is impacted by other factors, including warehouse layout and instance size. Instance size is used to provide a quantification of instance difficulty, and here we restrict it to number of orders, total number of products $|\mathcal{P}|$ and products which are to be assigned a location $|\mathcal{P}_s|$. The latter number, $|\mathcal{P}_s|$, is computed as 5-10% of $|\mathcal{P}|$ in the instance.

We proceed with a second experiments part, where we run the SLAP optimizer (Algorithm 1) on the industrial instances with and without the QAP model. For the experiments without the QAP model, $N = 1$ and lines 11 and 12 in Algorithm 1 are removed. This second part is carried out after suitable constants for N and λ values have been found on the L17_533.

In order to find such constants, we run the steps in Figure 5 for 10 N values ranged between 1 – 200 and 10 λ values set between 5 – 50% of $|\mathcal{P}_s|$. For the experiments to test N, we use $\lambda = 15\%$ of $|\mathcal{P}_s|$. For the experiment to test λ, we use $N = 50$. For the cost distance function Δ we use a scaled sigmoid, which is set to approach 1 when the ratio $f^*(x_i)/f^*(x_{i+1})$ exceeds 1.05. This means that sample x_{i+1} is unlikely to be accepted if its cost is 5% higher than that of x_i. For each instance, the global best OBP result is tracked and uploaded as the current best result. We refer to the documentation in L17_533 for further details. We use Intel Core i7-4710MQ 2.5 GZ 4 cores, 32 GB RAM, Python3, Cython and C.

7.2 Results

7.2.1 The impact of parameters N and λ on QAP predictive quality

Concerning N, we first observe that the average predictive quality of the QAP model is equivalent to the random baseline when $N = 1$ (Figure 6). We further observe that mean predictive quality rises steadily until N is 20, after which it tapers off.

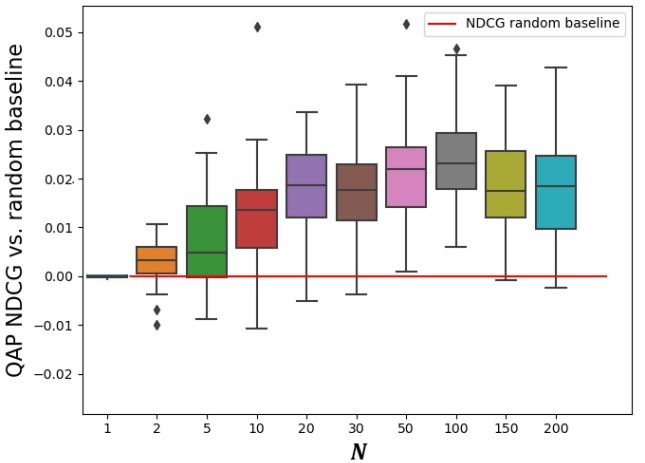

Figure 6: Boxplot showing number of samples (N) against QAP predictive quality. The red line denotes the NDCG random baseline. The box edges show the first and third quartiles of the data (Q1, Q3) and the whiskers show (Q1 – 1.5 * IQR, Q3 + 1.5 * IQR), where IQR is the Inter Quartile Range.

The result clearly shows that the QAP model is able to rank samples better than the random baseline (negative values imply the opposite). The positive initial trend could be impacted by the choice of ordinal relevance values $rel(\pi_{f(X)})$ for the NDCG computation (Section 7.1), which could favour the baseline for smaller N.

Concerning rate of change of new samples λ, the best results are achieved when it is set toward the lower end of the 5-50% range of $|\mathcal{P}_S|$ (Figure 7). This provides some validation for the use of a Metropolis algorithm, since it shows that a Markov Chain can be used to nudge samples closer towards lower costs. Otherwise, NDCG would be similar regardless of the x-axis in Figure 7. This result is in line with Oxenstierna et al., (2023), where a slightly stronger pattern is observed on the related TSP-based SLAP.

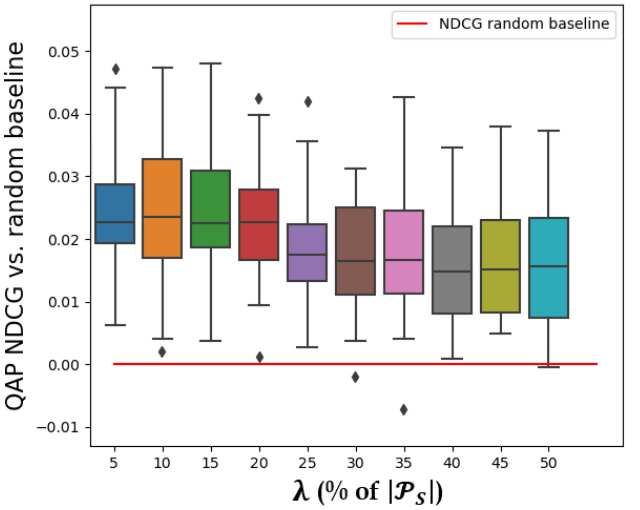

Figure 7: How much new samples are changed compared to previous samples (λ) against QAP predictive power.

7.2.2 The impact of other factors on QAP predictive quality

Results for all factors are shown in Table 1, 2 and 3 (Appendix). We find that QAP predictive quality decreases as instance size increases (Figure 8). This may be due to that the quality of $f^*(x)$ costs provided by the OBP optimizer decrease with instance size (they are sub-optimal, see Section 5.3), making analysis of results for larger instance classes more difficult in general. We find that the fraction of CPU-time required by the QAP model versus the OBP optimizer is between 0.006-0.019, or around 50-150 times faster. The difference is largest for the largest instances and smallest for the smallest instances (Table 2). We do not observe any relationship between QAP predictive quality and warehouse layout.

Overall, the result provides evidence that QAP approximations of OBP costs within an OBP-based SLAP optimizer may be justified. Its predictive quality may decrease with instance size, relative to the OBP optimizer (Figure 7), but its relative usage of CPU-time also decreases. Another way to visualize the performance difference between the QAP model and the random baseline is through a frequency distribution (Figure 9).

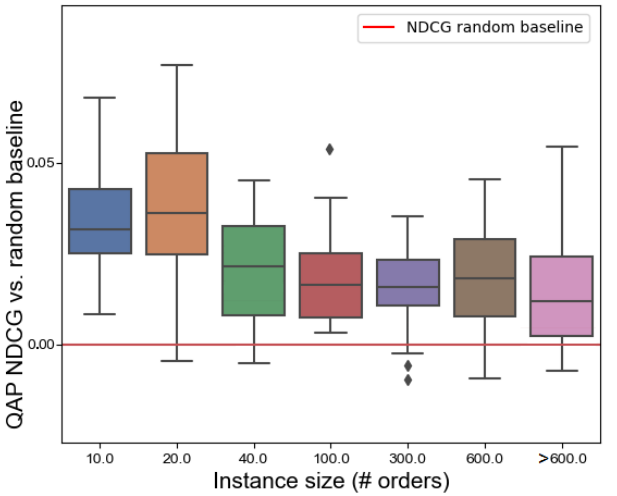

Figure 8: Instance size, in terms of number of orders, versus the predictive quality of the QAP model and the random baseline.

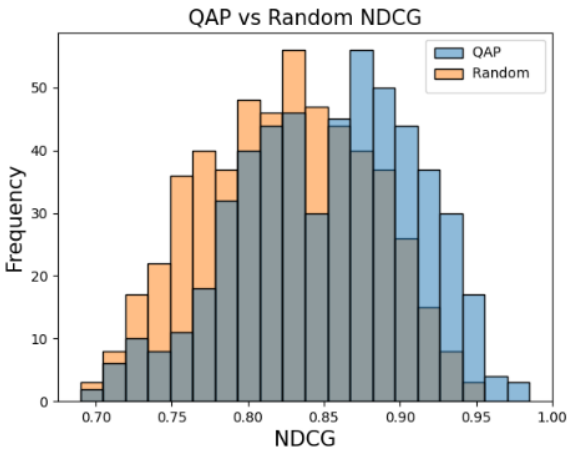

Figure 9: Frequency distribution of NDCG values (20 bins) from QAP and random ranking of samples when $N = 20$ and $\lambda = 10\%$ (of $|\mathcal{P}_S|$).

7.2.3 SLAP optimization with and without QAP approximation

We report results from running the QAP-OBP SLAP optimizer (Section 5) on the industrial dataset with and without the use of QAP approximations. Apart from general settings (Section 7.1), K is set to 10^8 and the algorithm is set to terminate after 60 minutes (which, given maximum OBP and QAP CPU-times, ensures iterations never exceed K). λ is set to 10% of $|\mathcal{P}_S|$ and $c_1 = 1$. N is set to 20, which means that the QAP model will have a relatively small impact on overall CPU-time. N could theoretically be set to a much larger number, but this may not necessarily yield better results. The QAP model in the form of Equation 5 likely needs to be further developed before its extended use can be motivated. One risk with setting N to a large number is that the SLAP optimizer will spend too much time in search regions with a low QAP cost, rather than in regions with a low OBP cost.

In Figure 10, we see that Algorithm 1, on average, improves cost by around 23% in 1 hour. Without QAP approximations, cost improves by around 17%.

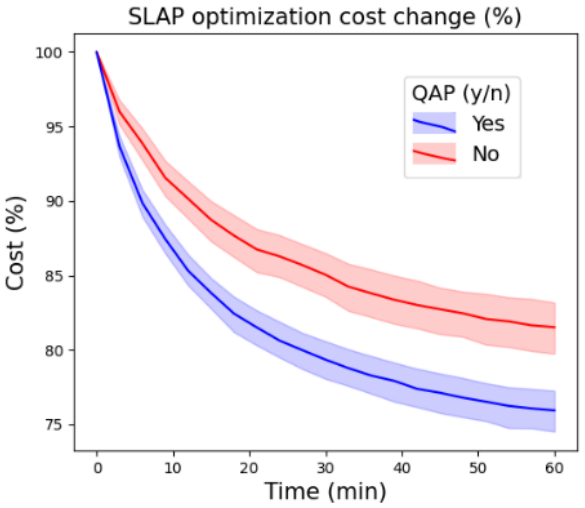

Figure 10: SLAP optimization cost improvements with and without the QAP model during 1 hour. The shaded areas denote 95% confidence intervals.

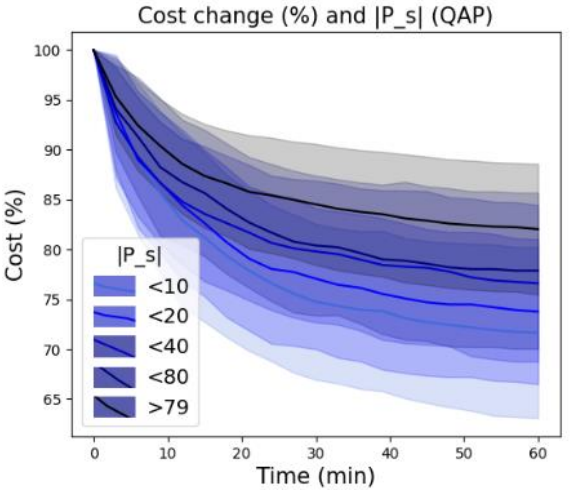

Figure 11: QAP-OBP SLAP cost improvement using QAP approximations for 5 categories of instance sizes (in terms of $|\mathcal{P}_s|$). Shaded areas denote data within 1 standard deviation.

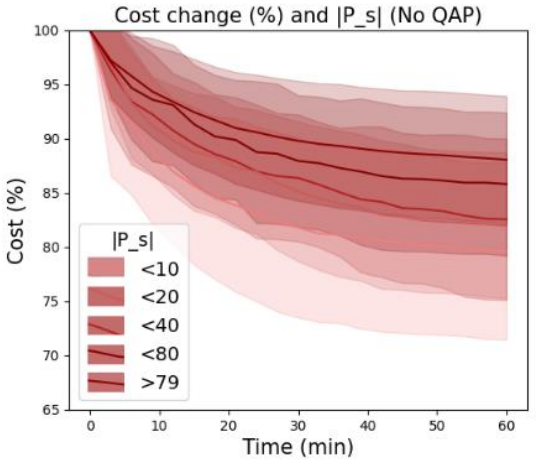

Figure 12: Same as Figure 11, but without using QAP approximations.

The size of the instances has a significant impact on computational efficiency. In Figure 11 and 12, we see that the impact of instance size, in terms of number of products that are assigned a location, $|\mathcal{P}_s|$, has a similar effect on computational efficiency regardless of whether the QAP is used. The stronger performance of the smaller instances can largely be attributed to more samples

being generated within the 60 minutes. On average, cost improvement continues throughout the time, which is explainable due to the large SLAP search space.

8. Conclusion

In this paper, we:

- formulate an optimization model for the Storage Location Assignment Problem (SLAP), where the costs of assignments are evaluated using Order Batching Problem (OBP) optimization.
- share generated SLAP test instances, with the goal to standardize formats and comparability between solution approaches.
- propose a Quadratic Assignment Problem (QAP) model to quickly approximate OBP costs in SLAP optimization. The QAP model is tested and tuned on the generated instances.
- propose a SLAP optimizer (QAP-OBP), which we test on industrial instances with a 1 hour optimization timeout.

Within the QAP-OBP optimizer, the QAP and OBP modules are utilized in a Metropolis algorithm, where samples are modified by a variable amount each iteration. The algorithm is nested such that OBP costs are only computed for samples with relatively strong QAP cost approximations.

In order to motive the use of the QAP model within the algorithm, experiments are first conducted to test its predictive quality against costs obtained by the OBP optimizer and a random baseline. Results show that QAP predictive quality is stronger than the baseline, and that they are around 50-150 times faster to compute than the cost obtained when using the OBP optimizer.

We then proceed to run the SLAP optimizer with and without the QAP approximations. We find that the optimizer performs better when using the QAP approximations, with cost improvements of around 23% after 1 hour. This result is in line with results in related work on SLAPs that are less difficult in some regards (for example concerning warehouse layouts), but more difficult in others (dynamicity or larger number of products).

For future work, the parameter which controls the number of samples that should be approximated by the QAP model for every OBP cost computation, N, could be tuned. The QAP computations could be significantly sped up by the use of parallelization and Graphical Processing Units (GPU), extending its utility within the SLAP optimizer for larger N. Also, alternative optimization approaches could be explored. These include meta-heuristic techniques such as Simulated Annealing or Particle Swarm Optimization. The QAP cost approximator could also be developed for a Machine Learning approach and used in a similar fashion as the weak estimators in boosting or aggregate bootstrapping. The factorial search space remains a fundamental problem for learning, however. Finally, we invite discussions into how to best represent SLAP features in public benchmark data and which features to choose for a standardized version of the problem.

Acknowledgements

This work was partially supported by the Wallenberg AI, Autonomous Systems and Software Program (WASP) funded by the Knut and Alice Wallenberg Foundation. We also convey thanks to Kairos Logic AB for software.

Compliance with ethical standards: Funding: This work was partially supported by the Wallenberg AI, Autonomous Systems and Software Program (WASP). Conflict of Interest: The authors declare that they have no conflict of interest. This article does not contain any studies with human participants or animals performed by any of the authors.

References

Abdel-Basset, M., Manogaran, G., Rashad, H., & Zaied, A. N. H. (2018). A comprehensive review of quadratic assignment problem: Variants, hybrids and applications. *Journal of Ambient Intelligence and Humanized Computing*, 1–24.

Aerts, B., Cornelissens, T., & Sörensen, K. (2021). The joint order batching and picker routing problem: Modelled and solved as a clustered vehicle routing problem. *Computers & Operations Research*, *129*, 105168. https://doi.org/10.1016/j.cor.2020.105168

Azadeh, K., De Koster, R., & Roy, D. (2019). Robotized and Automated Warehouse Systems: Review and Recent Developments. *Transportation Science*, *53*.

Bruch, S., Wang, X., Bendersky, M., & Najork, M. (2019). An Analysis of the Softmax Cross Entropy Loss for Learning-to-Rank with Binary Relevance. *Proceedings of the 2019 ACM SIGIR International Conference on the Theory of Information Retrieval (ICTIR 2019)*, 75–78.

Cao, Z., Qin, T., Liu, T.-Y., Tsai, M.-F., & Li, H. (2007). Learning to Rank: From Pairwise Approach to Listwise Approach. *Proceedings of the 24th International Conference on Machine Learning*, *227*, 129–136. https://doi.org/10.1145/1273496.1273513

Cardona, L. F., Rivera, L., & Martínez, H. J. (2012). Analytical study of the Fishbone Warehouse layout. *International Journal of Logistics Research and Applications*, *15*(6), 365–388.

Charris, E., Rojas-Reyes, J., & Montoya-Torres, J. (2018). The storage location assignment problem: A literature review. *International Journal of Industrial Engineering Computations*, *10*.

Christen, J. A., & Fox, C. (2005). Markov Chain Monte Carlo Using an Approximation. *Journal of Computational and Graphical Statistics*, *14*(4), 795–810.

Ene, S., & Öztürk, N. (2011). Storage location assignment and order picking optimization in the automotive industry. *The International Journal of Advanced Manufacturing Technology*, *60*, 1–11. https://doi.org/10.1007/s00170-011-3593-y

Fontana, M. E., & Nepomuceno, V. S. (2017). Multi-criteria approach for products classification and their storage location assignment. *The International Journal of Advanced Manufacturing Technology*, *88*(9), 3205–3216.

Freund, Y., Iyer, R., Schapire, R. E., & Singer, Y. (2003). An efficient boosting algorithm for combining preferences. *Journal of Machine Learning Research*, *4*(Nov), 933–969.

Freund, Y., & Schapire, R. E. (1996). *Experiments with a New Boosting Algorithm*.

Garfinkel, M. (2005). *Minimizing multi-zone orders in the correlated storage assingment problem*. School of Industrial and Systems Engineering, Georgia Institute of Technology.

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

Kallina, C., & Lynn, J. (1976). Application of the Cube-per-Order Index Rule for Stock Location in a Distribution Warehouse. *Interfaces*, *7*(1), 37–46.

Kofler, M., Beham, A., Wagner, S., & Affenzeller, M. (2014). *Affinity Based Slotting in Warehouses with Dynamic Order Patterns*. *Advanced Methods and Applications in Computational Intelligence*, 123–143.

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

Kübler, P., Glock, C. H., & Bauernhansl, T. (2020). A new iterative method for solving the joint dynamic storage location assignment, order batching and picker routing problem in manual picker-to-parts warehouses. *Computers & Industrial Engineering*, *147*, 106645.

Larco, J. A., Koster, R. de, Roodbergen, K. J., & Dul, J. (2017). Managing warehouse efficiency and worker discomfort through enhanced storage assignment decisions. *International Journal of Production Research*, *55*(21), 6407–6422. https://doi.org/10.1080/00207543.2016.1165880

Lee, I. G., Chung, S. H., & Yoon, S. W. (2020). Two-stage storage assignment to minimize travel time and congestion for warehouse order picking operations. *Computers & Industrial Engineering*, *139*, 106129. https://doi.org/10.1016/j.cie.2019.106129

Mantel, R., Schuur, P., & Heragu, S. (2007). Order oriented slotting: A new assignment strategy for warehouses. *European Journal of Industrial Engineering*, *1*, 301–316.

Oxenstierna, J., Malec, J., & Krueger, V. (2022). Efficient Order Batching Optimization Using Seed Heuristics and the Metropolis Algorithm. *SN Computer Science*, *4*(2), 107.

Oxenstierna, J., Rensburg, L. van, Stuckey, P., & Krueger, V. (2023). Storage Assignment Using Nested Annealing and Hamming Distances. *Proceedings of the 12th International Conference on Operations Research and Enterprise Systems - ICORES*, 94–105. https://doi.org/10.5220/0011785100003396

Oxenstierna, J., van Rensburg, L. J., Malec, J., & Krueger, V. (2021). Formulation of a Layout-Agnostic Order Batching Problem. In B. Dorronsoro, L. Amodeo, M. Pavone, & P. Ruiz (Eds.), *Optimization and Learning* (pp. 216–226). Springer International Publishing.

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

Reinelt, G. (1991). TSPLIB - A Traveling Salesman Problem Library. *INFORMS J. Comput.*, *3*, 376–384.

Rensburg, L. J. van. (2019). *Artificial intelligence for warehouse picking optimization—An NP-hard problem* [Master's Thesis]. Uppsala University.

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

van Ravenzwaaij, D., Cassey, P., & Brown, S. D. (2018). A simple introduction to Markov Chain Monte–Carlo sampling. *Psychonomic Bulletin & Review*, *25*(1), 143–154. https://doi.org/10.3758/s13423-016-1015-8

Wu, J., Qin, T., Chen, J., Si, H., & Lin, K. (2014). Slotting Optimization Algorithm of the Stereo Warehouse. *Proceedings of the 2012 2nd International Conference on Computer and Information Application (ICCIA 2012)*, 128–132. https://doi.org/10.2991/iccia.2012.31

Wu, X. (Bruce), Lu, J., Wu, S., & Zhou, X. (Simon). (2021). Synchronizing time-dependent transportation services: Reformulation and solution algorithm using quadratic assignment problem. *Transportation Research Part B: Methodological*, *152*, 140–179. https://doi.org/10.1016/j.trb.2021.08.008

Wutthisirisart, P., Noble, J. S., & Chang, C. A. (2015). A two-phased heuristic for relation-based item location. *Computers & Industrial Engineering*, *82*, 94–102. https://doi.org/10.1016/j.cie.2015.01.020

Yang, N. & others. (2022). Evaluation of the joint impact of the storage assignment and order batching in mobile-pod warehouse systems. *Mathematical Problems in Engineering*, *2022*.

Yingde, L., & Smith, J. S. (2012). Dynamic Slotting Optimization Based on SKUs Correlations in a Zone-based Wave-picking System. *IMHRC Proceedings*, *12*.

Zhang, R.-Q., Wang, M., & Pan, X. (2019). New model of the storage location assignment problem considering demand correlation pattern. *Computers & Industrial Engineering*, *129*, 210–219. https://doi.org/10.1016/j.cie.2019.01.027

Zhou, F., & De la Torre, F. (2016). Factorized Graph Matching. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *38*(9), 1774–1789. https://doi.org/10.1109/TPAMI.2015.2501802

Žulj, I., Glock, C. H., Grosse, E. H., & Schneider, M. (2018). Picker routing and storage-assignment strategies for precedence-constrained order picking. *Computers & Industrial Engineering*, *123*, 338–347. https://doi.org/10.1016/j.cie.2018.06.015

Appendix

NDCG flowchart: The below example shows how Normalized Discounted Cumulative Gain (NDCG) can be computed from input permutations (products to locations), approximated (f) and ground truth (f^*) values. Note that $f(X)$ denotes a sorting of X according to the cost valuation of elements in the *cost* step. Also note that relevance values can be formulated in several ways.

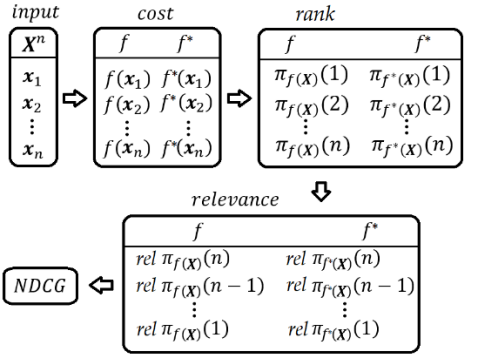

Figure 13: NDCG procedure flowchart.

Table 1: Summary of instances and results for different types of warehouse layouts. Also an aggregate of the results concerning the predictive quality of the QAP model.

		NoObstacles	SingleRack		NR1		L9_251
		Conventional	TwelveRacks		NR2		
Num Orders avg.		41	18	20	14	16	19	548
Num Procucts avg.		128	63	89	49	45	88	964
Num available locations % avg.		10	10	10	10	10	10	5
Num depots		2	2	2	2	2	2	2
CPU-time OBP model avg.		0.086	0.029	0.052	0.03	0.03	0.059	1.72
CPU-time QAP model avg.		0.001	0.0006	0.0008	0.0004	0.0004	0.0008	0.014
QAP NDCG difference between baseline and prediction								
Aggregates for n=(5, 10, 20, 30, 50, 100)								
Grand total avg.		0.03	0.02	0.03	0.02	0.01	0.04	0.01
Std. avg.		0.02	0.02	0.01	0.01	0.01	0.02	0.02
Vehicle Capacity Orders	2-4	0.01	0.02	0.03	0.01	0.02	0.05	0.01
	4-6	0.01	-	-	0.04	-0.01	0.02	0.02
	6-8	0.05	-	-	0.01	0.00	0.04	-0.01
	>8	0.03	-	-	-	-	-	0.01
Num orders	5-20	0.03	0.00	0.02	0.01	-0.01	0.05	-
	20-40	0.03	0.02	0.05	0.02	0.03	0.03	-
	100	0.03	-	-	-	-	-	0.03
	500	0.02	-	-	-	-	-	0.01
	>500	-	-	-	-	-	-	0.01
Num products	10-20	0.04	0.04	0.02	0.02	-0.01	0.06	-
	20-100	0.03	0.01	0.04	0.02	0.02	0.03	-
	100-500	0.01	-	-	-	-	-	0.02
	500-1000	-	-	-	-	-	-	0.04
	>1000	-	-	-	-	-	-	0.01

Table 2: Summary of results with regard to instance size. These results exclude instances with more than 435 products. All values are averages over instances with a certain number of products (num_products).

num_orders		λ (% of $\lvert\mathcal{P}_S\rvert$)			CPU-time OBP (s)	CPU-time QAP (s)	$\lvert\mathcal{P}_S\rvert$ (%)		NDCG performance versus baseline
num_products	num_samples		N				Impr. $f(x_1)$		
8	5.00	5021.74	22.08	54.88	0.031	0.001	10.00	16.07	0.031
14	9.13	11203.65	22.81	54.95	0.120	0.002	10.00	15.70	0.022
23	11.82	29037.20	25.98	55.00	0.023	0.001	10.00	15.40	0.037
32	15.45	21914.16	25.97	55.24	0.170	0.003	10.00	16.14	0.029
41	18.87	19745.97	26.00	55.00	0.165	0.002	10.00	15.80	0.028
50	23.46	19970.10	26.00	55.00	0.202	0.003	10.00	16.51	0.028
60	23.74	15425.63	25.86	55.03	0.311	0.005	10.00	13.11	0.047
69	25.53	11653.50	26.18	55.00	0.421	0.006	10.00	14.98	0.035
78	24.93	8744.61	26.04	55.30	0.404	0.005	10.00	16.60	0.026
87	29.16	10957.86	25.85	55.09	0.538	0.007	10.00	14.30	0.026
96	29.60	11323.38	26.07	54.52	0.524	0.006	10.00	12.00	0.026
106	24.17	10505.19	26.00	55.00	0.587	0.007	10.00	14.65	0.018
115	28.00	10280.07	25.96	54.97	0.631	0.007	10.00	11.35	0.037
124	33.00	18550.38	26.06	55.00	0.704	0.008	10.00	12.04	0.024
133	30.33	18496.34	26.05	54.51	0.568	0.006	10.00	12.76	0.024
142	33.00	31796.75	26.00	55.06	0.789	0.007	10.00	13.51	0.023
151	34.75	21580.00	25.85	55.41	0.864	0.008	10.00	9.18	0.029
161	38.67	13668.20	25.93	54.52	1.427	0.014	10.00	9.90	0.022
170	38.33	20419.26	26.00	55.24	1.014	0.009	9.12	11.57	0.021
179	32.75	10290.98	25.99	55.37	1.057	0.010	8.94	12.30	0.025
188	34.67	19598.84	25.96	55.00	1.528	0.013	8.56	13.03	0.020
197	32.00	9319.42	25.91	55.24	0.882	0.007	7.08	9.62	0.019
216	44.00	9140.06	26.00	54.60	1.751	0.014	6.98	10.40	0.010
225	37.50	11584.69	25.85	54.76	1.274	0.010	6.07	12.00	0.033
234	36.00	11511.74	26.00	55.22	2.173	0.016	4.99	11.80	0.011
252	39.00	9913.05	25.99	55.31	2.234	0.017	5.00	7.59	0.018
262	44.00	8519.34	25.85	54.80	2.642	0.020	4.94	7.14	0.017
289	39.50	9355.62	26.09	54.55	2.552	0.019	4.91	8.90	0.017
317	161.00	9672.15	25.80	55.00	2.427	0.017	5.00	6.57	0.026
326	162.00	8000.55	26.00	55.15	2.657	0.019	5.10	5.26	0.019
381	206.00	7495.94	27.19	55.17	2.592	0.018	5.05	6.00	0.015
390	207.00	8630.29	26.14	55.00	2.671	0.017	4.95	5.70	0.011
399	205.00	9639.09	26.45	54.63	2.393	0.015	5.00	4.44	0.014
418	210.50	8200.00	26.98	55.00	2.727	0.016	5.00	5.10	0.024
427	222.00	7546.00	26.67	54.55	2.726	0.016	5.07	4.80	0.013

Table 3: Results on 60 minute optimization runs.

num_products		CPU-time OBP Mean		CPU-time QAP Mean		Cost improvements (starting at 100%) at 20, 40 and 60 minutes using QAP (left) and not using QAP (right)					
P_s	num_orders	CPU-time OBP Std		CPU-time QAP Std							
11	104.46	22.61	0.05	0.02	0.001	0.001	80.09	74.55	71.02	86.69	80.80	76.76
16	150.10	40.29	0.09	0.08	0.001	0.001	82.91	77.77	73.96	86.01	81.61	79.00
21	178.05	57.62	0.13	0.09	0.002	0.001	81.02	75.66	73.54	87.83	82.27	79.42
26	244.86	56.95	0.20	0.08	0.002	0.001	83.63	75.76	72.53	88.94	84.60	80.82
31	278.14	144.71	0.32	0.21	0.002	0.000	82.76	77.81	74.20	88.75	83.42	81.98
36	362.20	196.95	0.34	0.09	0.003	0.002	85.37	78.61	76.09	89.66	88.05	85.39
41	444.35	239.29	0.48	0.21	0.004	0.001	82.33	74.37	70.19	91.87	83.43	81.79
46	428.29	269.85	0.49	0.12	0.004	0.001	83.03	80.15	77.92	89.33	84.83	82.50
51	470.26	276.92	0.69	0.31	0.005	0.004	85.30	81.27	79.40	88.88	85.68	83.37
56	539.26	338.38	0.75	0.36	0.005	0.002	83.23	80.41	75.53	89.01	86.01	83.77
61	546.20	331.63	0.70	0.21	0.006	0.003	86.85	83.15	80.87	88.72	85.01	81.98
66	640.93	416.04	1.09	1.13	0.007	0.003	83.96	78.54	75.55	90.22	86.51	85.09
71	673.02	427.68	1.05	0.28	0.007	0.003	82.98	77.39	74.44	89.51	86.43	85.00
76	699.86	380.58	1.09	0.34	0.009	0.006	85.05	80.76	78.98	87.42	84.84	82.05
81	787.48	405.63	1.16	0.30	0.009	0.004	82.96	78.28	75.77	92.84	89.89	88.58
86	770.31	434.21	1.13	0.41	0.010	0.004	87.32	82.69	79.06	89.71	85.25	82.91
91	919.05	607.36	1.68	0.62	0.012	0.006	90.08	89.08	85.66	89.68	83.39	82.30
96	912.42	515.95	1.53	0.48	0.012	0.004	86.84	83.13	81.29	89.15	84.57	82.55
101	917.49	566.34	1.64	0.82	0.012	0.005	84.79	82.14	80.11	92.82	88.33	85.33
106	949.39	567.17	1.66	0.51	0.013	0.006	85.80	82.62	79.47	88.27	84.05	81.20
111	1023.24	631.35	1.76	0.39	0.014	0.005	84.35	78.47	74.59	91.22	85.68	83.50
116	1043.54	559.44	1.92	0.76	0.014	0.005	85.56	81.51	80.51	92.50	87.85	86.51
121	1106.57	612.55	1.95	0.52	0.016	0.006	85.21	81.16	78.50	89.58	87.67	85.84
126	1151.81	610.52	1.99	0.42	0.017	0.006	87.32	84.13	83.13	91.30	87.66	84.82
131	1186.47	670.41	2.13	0.44	0.018	0.006	83.13	80.57	78.97	91.58	86.88	84.73
136	1265.07	741.22	2.26	0.47	0.020	0.008	88.33	83.93	81.70	90.50	86.06	83.93
141	1284.32	832.35	2.42	0.22	0.020	0.006	84.03	77.73	74.07	90.46	87.84	85.83
146	1348.17	736.48	2.43	0.70	0.021	0.003	84.03	75.75	74.30	89.77	87.39	86.31
151	1373.92	816.92	2.77	0.42	0.022	0.007	87.94	83.77	77.67	91.30	88.70	86.76
156	1352.87	797.75	2.48	0.52	0.021	0.006	87.99	86.20	84.51	90.02	88.48	87.43
161	1407.88	793.72	2.79	0.56	0.022	0.007	83.97	78.49	74.63	91.40	87.29	85.13
166	1462.75	720.25	2.96	1.20	0.023	0.005	81.70	77.92	75.87	89.61	85.19	84.19
171	1657.88	826.32	2.85	0.42	0.029	0.008	87.25	83.74	82.74	94.56	88.99	87.99
176	1565.06	821.50	2.95	0.39	0.027	0.007	86.95	84.11	80.61	95.82	90.60	89.60
181	1501.56	868.61	2.92	0.30	0.026	0.008	87.56	86.56	83.43	84.92	83.27	82.27
186	1596.48	801.25	3.82	3.67	0.027	0.006	84.91	80.92	77.31	92.59	91.55	87.35
191	1777.02	887.87	3.57	0.73	0.032	0.009	88.37	81.85	77.94	93.43	88.74	87.65
196	1562.60	794.37	2.75	0.19	0.028	0.001	87.99	86.99	85.99	81.05	77.50	76.50
201	1855.64	932.25	3.20	0.32	0.031	0.001	92.85	81.41	80.41	98.00	79.73	77.78
206	1808.08	920.67	3.41	0.41	0.032	0.009	90.44	82.14	81.14	91.85	87.87	86.87
211	1743.06	854.71	3.78	0.78	0.037	0.011	88.04	82.46	80.37	91.73	86.22	83.41
221	1782.87	883.77	4.17	0.79	0.033	0.008	83.68	77.54	74.95	91.47	84.01	81.03

Optimization of the Storage Location Assignment Problem using Nested Annealing

Johan Oxenstierna[1,4] [0000−0002−6608−9621], Louis Janse van Rensburg[3], Peter J.Stuckey[2] [0000000321860459], and Volker Krueger[1] [0000−0002−8836−8816]

[1] Dept. of Computer Science, Lund University, Lund, Sweden {johan.oxenstierna, volker.krueger}@cs.lth.se
[2] Faculty of Information Technology, Monash University, Australia
peter.stuckey@monash.edu
[3] Flux Robotics, Australia
louis@fluxrobotics.ai
[4] Kairos Logic AB, Lund, Sweden

Abstract. The Storage Location Assignment Problem (SLAP) has a significant impact on the efficiency of warehouse operations. We propose a multi-phase optimizer for the SLAP, where the quality of an assignment is based on distance estimates of future-forecasted order-picking. Candidate assignments are first sampled using a Markov Chain accept/reject method. Order-picking Traveling Salesman Problems (TSPs) are then modified according to the assignments and solved. The model is graph-based and generalizes to any obstacle layout in two dimensions. We investigate whether optimization speed-ups are possible using methods such as cost approximation, rejection of samples with low approximate cost and restarts from local minima. Results demonstrate that these methods improve performance, with total travel-cost reductions of up to 30% within 8 hours of CPU-time. We share a public repository with SLAP instances and corresponding benchmark results on the generalizable TSPLIB format.

Keywords: Storage Location Assignment Problem, Nested Annealing, Hamming Distances.

1 Introduction

The Storage Location Assignment Problem (SLAP) concerns the search for suitable locations for products in a warehouse. There exist dozens of proposed versions and optimization methods for the SLAP [5]. We work with a standard picker-to-parts scenario where racks and other obstacles can be laid out freely on a two-dimensional plane and where vehicles may start and end their paths at any location. In order to evaluate the quality of a location assignment, we combine two costs. The first cost consists of the travel distance needed to complete a given *picking-log*, i.e., a set of pick-rounds (sequences of product visits). A pick-round is equivalent to a *Steiner* Traveling Salesman

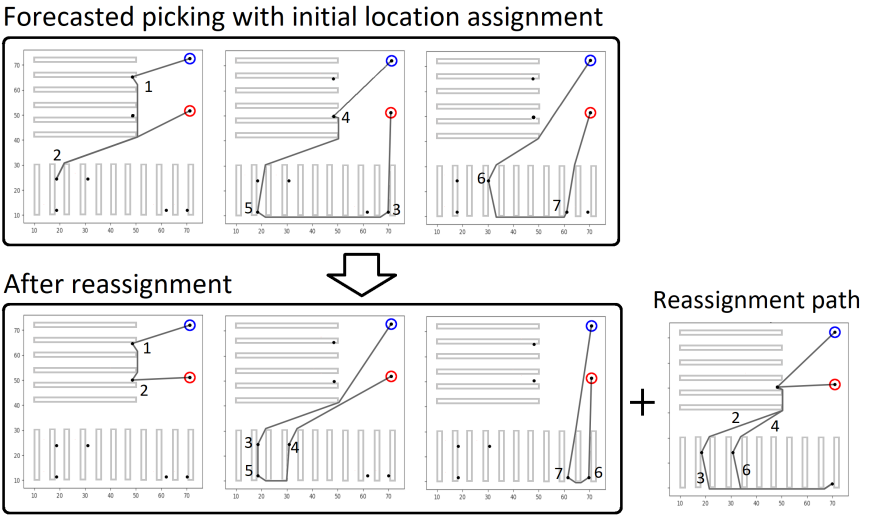

Fig. 1: Example of a SLAP with products enumerated 1-7 and an unconventional obstacle-layout [28]. The picking-log consists of three pick-rounds (TSPs) and their optimal solutions give the picking-log distance. The initial baseline assignment (top) has a longer picking-log distance compared to a candidate (sample) assignment (bottom left). In this example, the *reassignment path* needed to move the products according to the sample (bottom right), is longer than any possible savings concerning the picking-log (more pick-rounds are needed for savings).

Problem (TSP) [36], where the origin and destination locations may be different and where the same location may be revisited by one or several vehicles. We obtain the picking-log distance by solving all TSPs given a location assignment of products. The second cost is the travel distance needed to move the products such that the assignment is obtained, in a single *reassignment path*. We refer to this model as the TSP-based SLAP. A visualization of the TSP-based SLAP is provided in Figure 1.

In Section 2 we discuss strengths and weaknesses of proposed SLAP models in the literature. The TSP-based SLAP can be compared to the closely related Order Batching Problem (OBP)-based SLAP [26], where the picking-log is replaced by a set of orders (where an order is a set of products). The OBP-based SLAP requires the batching of orders into pick-rounds, as well as the subsequent TSP optimization of these pick-rounds, before quality of a proposed location assignment can be estimated. While the theoretical optimization gains may be higher in the OBP-based SLAP, its larger search space also adds significant challenges [33, 17].

Choice of SLAP-model is inevitably a trade-off between simplicity, on the one hand, and complexity, on the other. Regarding the former, there is a need in research to

discuss what a relatively simple and standardized version of the SLAP should entail, since there is little consensus on the matter [5]. Apart from order batching, examples of other optional features include various forms of dynamicity, warehouse layout, vehicle types, cost functions and reassignment scenarios. The TSP-based SLAP excludes order-batching and dynamicity and uses distance instead of more realistic but complex cost alternatives, such as time-based costs. Nevertheless, the TSP-based SLAP still poses a highly intractable problem. This is partly attributable to the reassignment distance. Hypothetically, more location reassignments are needed to obtain a lower picking-log distance, but more reassignments also lead to a longer reassignment distance. Thus, an equilibrium point between two adversarial problems must be found to attain a strong solution. One final and relatively novel feature of the TSP-based SLAP is that it does not assume a specific warehouse layout. Although this makes cost calculation more computationally expensive, by disallowing heuristics based on presumed rack-placements, it allows for a higher degree of generalization.

In Section 5 we introduce our optimization algorithm. It is based on Simulated Annealing and a Hamming-distance location-swap heuristic. Restarts from local minima, as well as two cost approximators, are investigated to potentially improve computational efficiency (cost improvement through CPU-time). One of the cost approximators is based on sub-optimal TSP optimization, while the other is based on a pick-frequency heatmap. In Section 6 we introduce three datasets, including a publicly shared benchmark instance set on the TSPLIB format [11], and corresponding computational results.

Our contributions are summarized as follows:

1. A SLAP optimizer using a novel version of the Simulated Annealing algorithm and experiments to test its computational efficiency.
2. Performance comparison of two cost-approximators utilized within the optimizer.
3. A publicly shared SLAP instance set on the TSPLIB format.

This paper is an extension of a ICORES-2023 paper [28]. Apart from a thorough revision of the text, the extension includes new data (dataset 3 in Section 6.3), a new cost approximator (Section 5.3), re-runs of previous experiments, as well as new experiments and results (Section 6 and Section 8).

2 Literature Review

In this section we discuss how the SLAP has been described and optimized in previous work. We particularly refer to the extensive literature review by Charris et al. [5]. There are several strategies for conducting a storage location assignment. These include *Dedicated*, *Random* and *Class-based*.

– *Dedicated*: The locations of products are assumed to never change. This strategy is suitable if the collection of products does not change much through time. If human

picking is used, this approach has the advantage that pickers can learn to associate products with locations, allowing for speed-ups in picking [43].
- *Random*: Products can be assigned any location in the warehouse. This is particularly suitable if the collection of products changes frequently.
- *Class-based (zoning)*: Each product is assigned a class and the warehouse is divided into zones. Each zone contains one or several classes of products. Class-based storage can incorporate dedicated and random strategies for certain zones and/or classes [23]

The quality of a location assignment can be modeled in several ways. In a human picking scenario, Larco et al. [18] show that there exists a relationship between the height at which products are placed and worker welfare. Worker welfare can be quantified by estimating parameters such as "ergonomic loading", "discomfort" or "expenditure of human energy" [5]. On a similar note and for autonomous vehicle or shuttle based storage and retrieval systems (AVS/R), there exists a model which has the objective to minimize "energy consumption" [2].

Another way to judge solution quality is through datamining, using computations such as support, confidence and lift [25]. These can also be used to propose concrete location assignments [14, 43]. Datamining is primarily focused on the statistical analysis of products and their relationships, but it is often combined with order-picking in a SLAP.

A third proposal studies the effect of traffic congestion. Bottlenecks can be caused if, for example, too many products with high pick-frequency are placed close to the depot. Lee et al. [19], propose Correlated and Traffic Balanced Storage Assignment (C&TBSA), a multi-objective SLAP model which aims to minimize traffic congestion, while also minimizing aggregate order-picking cost.

Order-picking has many variations, depending on obstacle layout, picking strategy and travel conventions [5, 23, 31, 41]. Concerning obstacle layout, we distinguish between two types: *Conventional* and *Unconventional*. In the conventional layout, warehouse racks are assumed to be organized in Manhattan style blocks with parallel aisles and cross-aisles. Conventional layouts are used in the majority of research on both order-picking and the SLAP [5, 15]. The unconventional layout includes the "fishbone" and "cascade" layouts [4, 5], as well as all other layouts that are not conventional. Regardless of layout, the picking path of a vehicle can be formulated as a Traveling Salesman Problem (TSP) where paths cannot intersect obstacles [12, 31]. For conventional layouts, the TSP is often optimized using S-shape or Largest-Gap algorithms [32]. For unconventional layouts, Google OR-tools or Concorde have been proposed [27, 31].

As mentioned in Section 1, the SLAP can be optimized as a joint problem with an Order Batching Problem (OBP). Proposals include Kübler et al. [17], Xiang et al. [40] and Maruyama et al. [24]. While these authors argue for this approach, arguments also exist against it, at least for certain settings [23]. One issue with the OBP-based SLAP is that the OBP is highly intractable in its own right, thus adding to the difficulties involved in optimizing an already challenging problem.

If order-batching is not included in the SLAP, heuristics such as Cube per Order Index (COI) [13] and Order Oriented Slotting (OOS) [23] have been proposed. COI assumes that products with relatively high pick-frequency and low volume should be placed close to depot. COI does not include associations between products and is therefore mainly suitable for pick-rounds with few picks, such as pallet-picking or certain AVS/R systems [2]. OOS, on the other hand, is specifically designed for scenarios where orders may contain more than one product. Mantel et al. [23] introduce a Quadratic Assignment Problem (QAP) heuristic which computes distances between products and the number of times products appear in the same order. The quality of a candidate location assignment can then be estimated using QAP. Similar methods to OOS are used by Žulj et al. [44], Fontana and Nepomuceno [8] and Lee et al. [19].

The SLAP usecase can be divided into two categories depending on the number of products that are to be moved. "Re-warehousing" is the case when a large proportion of products are moved, whereas a smaller proportion is moved in "healing" [14]. Movements can be conducted in many ways, each accompanied by a (re)assignment "effort". Kübler et al. [17] propose the following (re)assignment effort scenarios:

i Product A is moved to an unoccupied location.
ii Product A swaps location with product B.
iii Product A is moved to a location occupied by product B. Product B is moved to a new location. If there is a product C occupying the new location, the procedure continues until a final product is placed at an empty location.

Scenario i comes with the least (re)assignment effort and the effort grows through scenarios ii and iii. Apart from travel distance, time used for product removal/placement on shelves as well as administrative time, can be added to the effort computation [17].

When it comes to optimization algorithms for the SLAP, both exact and non-exact methods have been proposed. The exact algorithms include dynamic programming, branch and bound algorithms and Mixed Integer Linear Programming (MILP) [5]. The SLAP search space is often reduced in scope when exact solutions are sought. These include restricting the number of locations [38], number of products [9, 21] or by only working with conventional warehouse layouts [3].

More commonly, non-exact heuristic or meta-heuristic algorithms are used. Proposals include Particle Swarm Optimization (PSO) [17], Genetic and Evolutionary Algorithms [7, 19, 20] and Simulated Annealing [14, 43]. The SLAP is often optimized in multiple phases using these methods. One example is to first generate candidate products for location assignments using datamining, and to then evaluate various candidate assignments using order-picking optimization [14, 39].

It is challenging to judge optimization results in previous work due to the multitude of variations in SLAP models [5]. For results including reassignment costs, conventional warehouse layouts, dynamic picking patterns and meta-heuristic optimization, Kofler et al. [14] report best savings around 21%. In a similar scenario, Kübler et al. [17] report best savings around 22%. Excluding reassignment costs, Zhang et al. [43]

report best savings around 18% on simulated data with thousands of product locations. In similar settings, Trindade et al. [35] report best savings around 33%, using a multi-phase optimizer, and Chiang et al. [25] report best savings around 13% using datamining heuristics and integer programming.

3 Simulated Annealing

Simulated Annealing, which draws inspiration from the annealing process in metallurgy [14], has a useful analogue with SLAP optimization: A poor storage assignment can be viewed as more energetic as it leads to more travel for picking in the warehouse. As the SLAP is optimized, products are reassigned to new locations using a decreasing temperature. As temperature cools, products become fixed in a lower energy state where picking travel costs are reduced. There are many complicating factors in the SLAP which can prevent a smooth decent toward an improved storage assignment, however. In the remainder of this section, we describe the Simulated Annealing algorithm and how it may be modified to help attain stronger results in the SLAP.

A key component in Simulated Annealing (Algorithm 1) is the *sample* function. In each iteration i, sample x_{i+1} is drawn based on a desired distance to sample x_i. This distance is computed using the probability distribution $q(x_{i+1}|x_i)$, without involving the cost of the samples (henceforth we refer to this as the *feature-distance*). The q distribution is often chosen to be Normal, so that the distance between x_i and x_{i+1} is low with high probability [22]. The $cost^*$ function computes/retrieves the cost (f^*) of the new/previous sample (the first sample is retrieved from memory after the first iteration). The accept probability α^* is based on a *cost-distance* function Δ (which outputs a negative value if the new cost is lower than the previous) and a decreasing temperature function T [29]. Functions for q, T and Δ are further discussed in Section 5.

Simulated Annealing is a type of Markov Chain Monte Carlo (MCMC) method and one advantage of this type of method is that its bias-variance tradeoff can be tuned using relatively few parameters [10]. A known disadvantage is that only two samples are stored in memory at any given time, which risks leading the Markov Chain to convergence on weak local minima [22]. Several methods have been proposed to reduce this risk, including mode-jumping [34], Nested Annealing [29] and Basin Hopping [37]. These methods split the search space into regions which are then subjected to local search. Another method is the Restart Strategy (SARS), which restarts the search from a random new sample whenever a "non-improving" local minimum is found [42].

Simulated Annealing can be modified to include a cost approximator, f, which provides fast cost estimates of f^*, to potentially increase computational efficiency. Christen and Fox [6] propose to use f to reject new samples that are unlikely to yield an improvement in f^* over the previous sample. The common MCMC *accept* method is accordingly split into two parts: *Promote* (f^* cost evaluation for a sample with a strong f) and *accept* (update x_i for the next iteration to be a sample with a strong f^*).

Algorithm 1 Simulated Annealing

x_i: Sample (an assignment).
$f^*(x_i)$: Ground truth cost of sample x_i.
q: Feature-distance function.
Δ: Cost-distance function.
N: Number of iterations.
T: Temperature function.
x_1: Initial sample (baseline).
for $i = 1, ..., N$ **do**
    $t \leftarrow T(i)$
    $x_{i+1} \leftarrow sample(q(x_{i+1}|x_i))$
    $f^*(x_i), f^*(x_{i+1}) \leftarrow cost^*(x_i, x_{i+1})$
    $\alpha^* \leftarrow exp(-c_1\Delta(f^*(x_{i+1}), f^*(x_i))/t)$
    $u \leftarrow \mathcal{U}(0,1)$ // random uniform
    if $u < \alpha^*$ **then** // sample *accepted*
        $x_i \leftarrow x_{i+1}$
    end if
end for

In our optimization algorithm (Section 5), we utilize this concept and split Simulated Annealing into *promote* based on a fast and less accurate costs computed in f, and *accept* based on slow and more accurate costs computed in f^*.

4 Problem Formulation

4.1 Objective Function

The objective in the TSP-based SLAP is to minimize the aggregate travel distance to:

1. Complete a given picking-log (a set of pick-rounds) $\mathcal{B}$.
2. Carry out any proposed location reassignments in a single reassignment path $\mathcal{R}$.

Each pick-round $b \in \mathcal{B}$ is a list of products. The set of all locations (including pick-locations, origin and destinations and obstacle corners in 2D Cartesian space) is denoted $\mathcal{L}$ and the set of all pick-locations is denoted $\mathcal{L}(\mathcal{P})$. The set of all products in $\mathcal{B}$ is denoted $\mathcal{P}$. Each product $p \in \mathcal{P}$ is defined as a tuple including a unique key (Stock Keeping Unit), a pick location $l(p) \in \mathcal{L}(\mathcal{P})$ and a positive pick frequency count $F(p)$. Each pick location is a tuple consisting of a unique key, a capacity and a location (represented as a node in a graph). A product is located at strictly one location and a location stores strictly one product. A product is allowed to move from its initial location to a new one as long as the new location's capacity is not exceeded.

A SLAP solution candidate (also referred to as sample or assignment) is represented as permutation vector $x \in X$, where the elements are enumerated product keys and the indices are enumerated locations. For an example warehouse with 3 locations, sample $x = (2,1,3)$ means that product 2 is assigned location 1, 1 assigned 2 and 3 assigned 3. Each sample x contains positive permutation integers in range 1 to m, $2 \leq m \leq |\mathcal{P}|$ and each permutation x has ground truth cost $f^*(x)$. m denotes the number of products that are subject to location change, and it does not necessarily have to be equal to the number of products in the warehouse, but could instead be manually set to limit the search space. Sample x_1 represents the baseline product location assignment (the initial locations of the products). In order to evaluate performance in optimization experiments, costs $f^*(x_2), f^*(x_3), ..., f^*(x_N)$ are compared against $f^*(x_1)$.

The objective in the TSP-based SLAP, is to find a sample assignment x such that picking-log cost $\sum_{b \in \mathcal{B}} D(b)$ and reassignment cost $D(\mathcal{R})$ are minimized:

$$\underset{x}{\operatorname{argmin}}((\sum_{b \in \mathcal{B}} D(b)) + \lambda D(\mathcal{R})) \tag{1}$$

Constant λ is used to weigh the two cost terms. Below we show how the picking-log and reassignment costs are computed using Euclidean distances.

4.2 Picking-log distance

The cost of all pick-rounds in picking-log $\mathcal{B}$ is computed using distance $\sum_{b \in \mathcal{B}} D(b)$. $D(b)$ is the distance of the solution to the Traveling Salesman Problem (TSP) represented by product locations in b:

$$D(b) = d_{l(origin_b),l(p_1)} + d_{l(p_{|b|}),l(destination_b)} + \sum d_{l(p_i),l(p_j)}, j = i+1, 0 < i < |b| \tag{2}$$

where $d_{l(p_i),l(p_j)}$ denotes the distance between the locations of $p_i, p_j \in b$, and where $d_{l(origin_b),l(p_1)}$ connects an origin location and $d_{l(p_{|b|}),l(destination_b)}$ a destination location to the path. The location of a product $l(p_i)$ is obtained from an index in the location assignment sample x. This index is stored for each product and updated whenever it changes location. We assume shortest distances and corresponding shortest paths (needed if path visualization is sought) between pairs of locations are queryable from Random Access Memory (RAM). All shortest paths and distances are pre-computed using the Floyd-Warshall graph algorithm, using a warehouse digitization process beyond the scope of this paper [31]. This process includes capability for uni-directed and mixed graphs, but in this paper we only work with bi-directed graphs (meaning that the distance between two locations is equal in both directions). We allow the origin and destination locations in the pick-rounds to be any locations in $\mathcal{L}$ (concerning TSP optimization, this is sometimes referred to as a Multi-Depot TSP or Dial-a-ride Problem). In Section 5 we describe how TSP optimization works for the multi-depot requirement.

4.3 Reassignment distance

Reassignment path $\mathcal{R}$ and its distance $D(\mathcal{R})$ is based on direct and indirect exchange scenarios (scenarios ii and iii in Section 2) with the following assumptions: Since there are an equal amount of products and locations in the SLAP, scenarios ii and iii represent a bijective relationship between products and locations. When products change locations, the bijection can take three forms: Direct exchange, e.g. $x_1 = (1,2)$ to $x_2 = (2,1)$ (product 2 goes to location 1 and 1 goes to 2), indirect exchange, e.g. $x_1 = (1,2,3)$ to $x_2 = (3,1,2)$ (1 goes to 2, 2 goes to 3 and 3 goes to 1), or a combination of both. We also assume that the operation to change locations of products, using direct and indirect exchanges, can be carried out by a single vehicle traveling along a single path through the warehouse, without intermediate stops at the depot. Algorithm 2 shows how this single reassignment path can be constructed, just from information in the initial assignment x_1 and a subsequent sample x_{1+i}, generated during optimization iteration $i < N$.

Algorithm 2 Reassignment Path and Distance

```
1: x_1: Initial assignment sample (baseline solution).
2: x: Sample obtained during SLAP optimization.
3: x_m ← copy(x)
4: D(R_best) ← ∞
5: for j = 1,...,K do // iterations.
6:     R ← list()
7:     while x_m not_empty do
8:         r ← list()
9:         while not_completed(r) do
10:            add_to_subcycle(r, x, x_m, x_1)
11:        end while
12:        R += r
13:    end while
14:    shuffle_and_flatten(R)
15:    D(R_best) ← update_best(R, R_best)
16: end for
```

r denotes a sub-cycle of locations (a sequence that starts and ends at the same location). The *add_to_subcycle* function has two cases:

1. If the r sequence is empty, a random new element is removed from x_m and its initial location (the index for that product in x_1) is added to r.
2. If r is not empty, the new location of the last added product in r is first found in x and added to r. The product located at that "next" location is found in x_1, matched in and then removed from x_m.

If the added location to r is equivalent to the first one in r, the sub-cycle is completed and r is added to $\mathcal{R}$. After x_m is emptied, $\mathcal{R}$ is first randomly shuffled and then flattened (the inner lists of sub-cycles are converted into a single list). The distance $D(\mathcal{R})$ is then computed as the sum of all location to location distances in $\mathcal{R}$, plus the distance from an origin depot location to the first location in $\mathcal{R}$ and the last location in $\mathcal{R}$ to a destination depot location. At each iteration, the $update\_best(\mathcal{R}, \mathcal{R}_{best})$ function updates the lowest minimum found by comparing distance $D(\mathcal{R})$ and distance $D(\mathcal{R}_{best})$. For Algorithm 1 and our modifications to it in Algorithm 3, $D(\mathcal{R})$ is included in the $cost^*$ and $cost$ functions.

In summary, reassignment path $\mathcal{R}$ is a solution to a constrained, linked-list TSP where a product is dropped off and another product picked up at each location. The vehicle conducting the reassignment path is assumed to be able to carry the whole quantity (frequency $F(p)$ in our case) of any single product located at any single location. A model of the reassignment path involving vehicle-capacities, enforcing return trips to depot when a product quantity exceeds vehicle capacity, is left for future work.

5 Optimization Algorithm

5.1 Assignment sampling using Markov Chain Monte Carlo (MCMC) and Hamming Distances

As described in Section 3, the Simulated Annealing algorithm includes two distributions to describe the amount of distance between samples x_i and x_{i+1}: Feature-distance q and cost-distance Δ. For sampling to be effective, there should exist some degree of proportionality between these two distributions. If the feature-distance between x_i and x_{i+1} is relatively low, the distance between costs $f^*(x_i)$ and $f^*(x_{i+1})$ should also be relatively low. The cost-distance in a SLAP is in the domain $\mathbb{R}^+$, as it represents Euclidean travel distances in the warehouse. The feature-distance between two samples is represented by the difference between two assignments. We hypothesize that the feature-distance can be computed using a Hamming distance heuristic. Hamming distance is a count of the number of non-identical elements between two permutation vectors (which are equivalent to assignments) [30]. The following sampling distribution is then proposed to utilize this Hamming distance (based on bounds proposed by Christen and Fox [6]):

$$q(x_{i+1}|x_i) = e^{-CH_d(x_i,x_{i+1})^P} \tag{3}$$

where C and P are hyperparameters in $\mathbb{R}^+$, and H_d denotes Hamming distance. We propose to use this sampling function within Algorithm 1. Below we propose methods which may improve computational efficiency (cost reduction through CPU-time) of Algorithm 1.

5.2 TSP optimization and cost caching

We utilize two TSP optimizers to compute the picking-log distances of assignment samples. For optimal TSP solutions we use the Concorde TSP solver [5] [1]. For approximate TSP solutions we use the OR-tools TSP optimization suite[6] [16]. In order to limit the CPU-time of OR-tools, we use the *solution_limit* parameter. For both these TSP optimizers, multi-depot scenarios are handled by modifying the input distance matrix with a dummy location whose distance is zero to the origin and destination, and whose other distances are set to infinity.

Before we apply TSP optimization to compute picking-log distance of an assignment sample, we reduce CPU-time through a filtering technique. Given the usage of sampling distribution q (Equation 3), we note that many pick-rounds will often not contain products that had their location changed. For example, assume we start with assignment $x_1 = (2,1,3)$ and two pick-rounds in the picking-log, one containing products 1 and 2 and the other containing product 3. Picking-log distance is then computed by TSP-optimizing the two pick-rounds (to keep the example small, we disregard the fact that TSP optimization only yields savings for longer pick-rounds). Assume we then swap locations of products 1 and 2: $x_2 = (1,2,3)$. Since product 3 remains at its initial location, there is no need to re-optimize the pick-round which contains that product. To enable this reduction of redundant TSP-optimization, we cache the TSP costs (both optimal and approximate) of any pick-round once computed. These costs are then queried for the pick-round until one or several product locations are changed, at which point the TSP gets re-optimized and the costs updated (only after promotion in the case of f^*).

5.3 Heatmap-based approximation

In order to motivate SLAP optimization, results need to be as interpretable and visually representable as possible. One problem with TSP optimization within a SLAP is that results cannot be easily visualized. Visualizing TSPs entails showing them before and after SLAP optimization. Figure 1 and Figure 8 (Appendix) are examples. Interpretation of these types of figures becomes very challenging when the picking-log contains hundreds of pick-rounds.

One possible way with which to visualize SLAP optimization in a single figure, is a *heatmap*. Figure 2 is an example which shows number of picks at 2700 locations (several locations share a single cell in the heatmap). The lower picture shows the result after SLAP optimization. To achieve this movement of the "hotter" products closer to depot, a dot product is first computed between the pick frequency count of each product $F(p)$ and the distance of their locations from an origin location and to a destination location:

[5] https://math.uwaterloo.ca/tsp/concorde/downloads /downloads.htm, collected 27-05-2022.

[6] https://developers.google.com/optimization/routing/tsp, collected 12-06-2022.

$$\sum_{i=1}^{|\mathcal{L}|} F(p_i)(d_{l(origin),l(p_i)} + d_{l(p_i),l(destination)}) \tag{4}$$

Location swaps are then conducted based on this dot product. For the heatmap example in Figure 2, 200 swaps were conducted to achieve a reduction of cost, according to Equation 4, of around 35%. Apart from the visual interpretability, an additional advantage of using Equation 4 is that it is very fast to compute. In Section 6 we conduct an experiment to investigate whether there is any correlation between this approximation and optimal TSP cost. The predictive quality of Equation 4 is likely weak, but if CPU-time is low enough it could still outmatch the alternative f approximation achieved by the OR-tools TSP optimizer. Note that this approach only works for cases when all pick-rounds in the picking-log use the same origin and destination location.

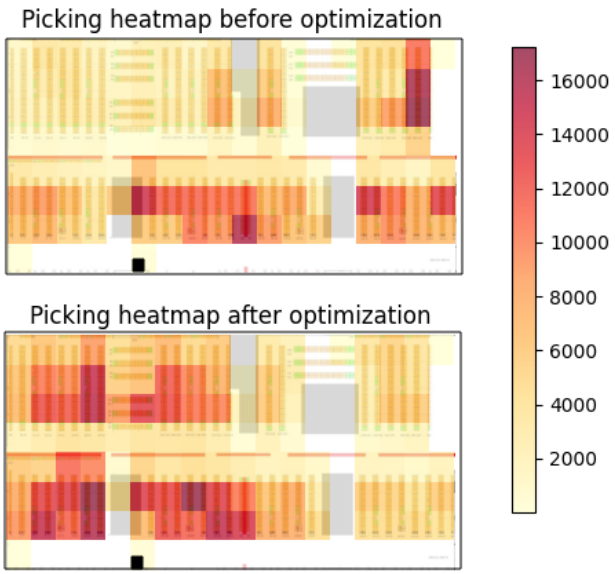

Fig. 2: Heatmap of picking in a warehouse with a single depot location (the black square). The colorbar shows how many picks occur within a given cell.

5.4 Nested Annealing

In Section 3 we suggested that the computational efficiency of Simulated Annealing (Algorithm 1) can be increased if there exists a function f which can quickly esti-

mate f^* . We then proceeded to propose two suggestions for such an f: One using sub-optimal TSP optimization (OR-tools), and one using a heatmap based approximation. In Algorithm 3, we show how either of these can be utilized within a modified Simulated Annealing algorithm:

Algorithm 3 Nested Annealing (based on computational efficiency in cost estimation

1: x_i: Sample (candidate solution).
2: $f(x_i)$: Less accurate fast cost estimate.
3: $f^*(x_i)$: More accurate slow cost estimate.
4: q: Feature-distance function.
5: Δ: Cost-distance function.
6: α: Probability that sample x_{i+1} is *promoted*.
7: α^*: Probability that sample x_{i+1} is *accepted*.
8: N: Number of iterations.
9: T: Temperature function.
10: x_1: Initial assignment sample (baseline).
11: **for** $i = 1, ..., N$ **do**
12: $t \leftarrow T(i)$
13: $x_{i+1} \leftarrow sample(q(x_{i+1}|x_i))$
14: $f(x_i), f(x_{i+1}) \leftarrow cost(x_i, x_{i+1})$
15: $\alpha \leftarrow exp(-c_1\Delta(f(x_{i+1}), f(x_i))/t)$
16: $u \leftarrow \mathcal{U}(0,1)$ // random uniform
17: **if** $u < \alpha$ **then** // sample *promoted*
18: $f^*(x_i), f^*(x_{i+1}) \leftarrow cost^*(x_i, x_{i+1})$
19: $\alpha^* \leftarrow exp(-c_2\Delta(f^*(x_{i+1}), f^*(x_i))/t)$
20: $u \leftarrow \mathcal{U}(0,1)$
21: **if** $u < \alpha^*$ **then** // sample *accepted*
22: $x_i \leftarrow x_{i+1}$
23: **end if**
24: **end if**
25: **end for**

After a sample x_{i+1} is generated, its *cost* is estimated using f. If the sample passes the *promote* filter on Line 17, $cost^*$ is computed using f^*. Note that the *cost* and $cost^*$ functions include reassignment distance $D(\mathcal{R})$ (Algorithm 2). Since Algorithm 2 does not guarantee optimality for $D(\mathcal{R})$, $cost^*$ does not guarantee optimality either, and hence we refer to f^* as "more accurate" rather than optimal. Hyperparameters $c_1, c_2 \in \mathbb{R}^+$ may be set differently. Christen and Fox [6] suggest setting $c_1 > c_2$ so that the promotion of a sample is less likely than the acceptance of a promoted sample. For the temperature function T we use a shifted and scaled reverse sigmoid (decreas-

ing) that gives temperatures in range $[1,0]$. For the cost-distance function Δ we use a shifted and scaled sigmoid that gives values in range $[0,1]$. Nested Annealing was first introduced by Rajasekaran and Reif [29], but they do not use cost approximation and base the nesting on variable set temperatures in local search regions. Algorithm 3 offers an alternative nesting strategy, based on a trade-off between predictive speed and accuracy.

5.5 Restarts

Due to the large search space of the SLAP, the MCMC sampling function $x_{i+1} \leftarrow sample(q(x_{i+1}|x_i))$, may benefit from occasional *restarts* (Section 3). Yu et al. [42], propose restarts from randomly generated samples. Their test-problems do not include reassignment distances, however, and in the SLAP, randomly generated samples can be expected to have a significantly higher cost than x_1 due to reassignment distance $D(\mathcal{R})$. As a solution to this problem, we instead propose restarts from local minima. The best minimum found through optimization is denoted x_{best} and it is used as restart sample with an increasing probability. Forcing restarts from x_{best} is motivated because its local neighbourhood cannot be extensively searched for in any but the smallest SLAP test-instances. A second minimum is denoted x_{lowR} and it is used as a restart sample with a decreasing probability. Forcing restarts from x_{lowR} is designed to target a low reassignment distance $D(\mathcal{R})$. The first such local minimum is $x_{lowR} = x_1$, whose $D(\mathcal{R}) = 0$. $x_{lowR} = x_1$ can be assumed to be a strong local minimum, due to its lack of reassignment distance, but after $f^*(x_1)$ has been beaten by $f^*(x_{1+i})$, x_{lowR} is updated at regular intervals to a previously generated sample which has a relatively low f^* cost and $D(\mathcal{R})$. In Section 6 we propose probability distributions for x_{best} and x_{lowR}, as well as optimization results with and without the use of restarts.

6 Experiments

6.1 Overview

We carry out experiments to investigate the following topics with regard to computational efficiency (cost reduction through CPU-time), in chronological order specified below:

1. Utility of *Hamming-distance* based sampling (q).
2. Utility of *restarts*.
3. Comparison of two cost approximators for use within Algorithm 3.
4. Comparison of Algorithm 1 and Algorithm 3 (using best settings from 2 and 3).
5. Other features (such as layout and number of products and pick-rounds).

All experiments are carried out using Intel Core i7-4700MQ, 2.40GHz, 4 cores and Python3 (with heavy use of Cython) and C.

6.2 Parameters

For all experiments, the number of products open for location reassignment (m) is set to be equivalent to the number of products in the test-instance. The number of reassignment path optimization iterations (K in Algorithm 2) is set to 300. After optimization has completed, the reassignment path is re-optimized with K set to 10000. The accept probability computation is set to be equivalent between Algorithm 1 and 3 ($c_2 = 1$ and equivalent Δ and T functions). The Δ function is set to approach 1 when the ratio of the distance between a new sample and a previous sample exceeds 1.05: This means that if a new sample has a distance 5% higher than the previous sample, it is unlikely to be promoted and/or accepted. c_1 in Algorithm 3 is set to 2, which makes it more difficult for a sample to be promoted than accepted once promoted. The reverse sigmoid probability distribution q, which gives the number of location changes between a new and a previous sample, is set to approach zero when number of location changes exceeds 20. For all experiments where a restart strategy is used, sample x_{i+1} can be built from either x_i, x_{best} or x_{lowR} (Section 5). The probability to pick one of the latter two is governed by a sigmoid and reverse sigmoid, respectively, with probabilities in range $[0, 0.2]$ and $[0.2, 0]$, stretched over N iterations. In all iterations where neither x_{best} nor x_{lowR} is picked, x_i is used (no restarts). λ and N are set depending on the dataset.

6.3 Datasets

The following three datasets are used:

1. 266 TSPLIB instances[7] modified for the SLAP and shared in a public repository[8]. These instances include 6 different types of warehouse layouts (including one with no obstacles). The number of products open for location reassignment vary between 5-427 in these instances. The initial locations for all products (baseline assignment x_1) in these instances is selected using a random uniform distribution. Solution proposals are uploaded for each of these instances using Algorithm 3 after a maximum of 20000 iterations (N). Experiments to test utility of Hamming distances and restarts are conducted on this dataset. λ is set to 1 for experiments on this dataset.
2. Data from a real warehouse with a conventional layout. The provided picking-log includes 260 unique products and 260 product locations. There are 200 pick-rounds and most products are picked in several pick-rounds. The experiments where Algorithm 1 and 3 are compared are run on this dataset. Algorithm 1 and 3 are run 10 times each on this dataset, with varying random seeds and a maximum CPU-time set to 8 hours. λ is set to 1 for experiments on this dataset.
3. Data from a real warehouse with an unconventional layout. Specific to this dataset is that there is only a single origin/destination and that some products are not

[7] https://github.com/johanoxenstierna/OBP/instances, collected 19-10-2022.

[8] https://github.com/johanoxenstierna/L40_266, collected 14-11-2022.

located in the warehouse apriori. These products are assigned random initial locations. There are also more locations than products in this dataset. The empty locations are utilized in optimization by placing a mock product at each of them. By flagging these products, they can be excluded from cost computation, while remaining open for product locations swaps. This dataset also contains longer pick-rounds than the other two (with an average of 29 picks per pick-round). The experiments where the two cost approximators are compared are conducted on this dataset, using a maximum CPU-time of 4 hours. λ is set to 0 for experiments on this dataset: This removes the reassignment distance and thus ensures that the two approximators can be compared against an optimal f^*.

In all three datasets, the capacity of all locations is assumed to be identical, meaning that any product can be placed at any location. We compare costs of samples against the baseline x_1, where each product is fixed to its initial location, where optimal picking costs are computed in $D(\mathcal{B})$ and where $D(\mathcal{R}) = 0$.

6.4 Experiment results

Utility of Hamming-distance based sampling Results show that many location reassignments are needed to reach the best reductions in travel cost (Figure 3). Also, more reduction in cost is achieved when the Hamming distance (number of location changes) between a previous sample and a new one is relatively low (Figure 4). On average, the cost of sample $f^*(x_{i+1})$ is more reduced compared to a previous sample $f^*(x_i)$ if fewer location changes are attempted. This result empirically validates the Hamming distance distribution $q(x_{i+1}|x_i)$ and its bias toward conducting fewer location changes at each step in the Markov Chain (Equation 3).

Utility of *Restarts* Results with and without *restarts* (Section 5) are shown in Figure 5. Given the same amount of optimization iterations ($N = 30000$) on dataset 2, the best results for both Algorithm 1 and 3 are obtained using restarts. Restarts enforce revisits to local minima with relatively short total travel costs f^* or reassignment costs $D(\mathcal{R})$ (Section 5). Since fewer reassignments mean that fewer pick-rounds contain products whose locations change, total TSP optimization CPU-time is significantly lower when restarts are used. This is achieved by the caching of TSP costs (Section 5). Furthermore, few reassignments mean that the optimization of the reassignment path requires less CPU-time to reach a relatively strong solution. As can be observed, Algorithm 1 and 3 without restarts (lighter blue and green) quickly jump up in cost. This is mainly attributed to the relatively low cost in assignment x_1, where $D(\mathcal{R}) = 0$, which is never revisited once stepped away from and never improved on (without restarts).

Comparison of the two cost approximators for Algorithm 3 Results on dataset 3 are summarized in Table 2. We first study the coefficient of determination R^2 (goodness of

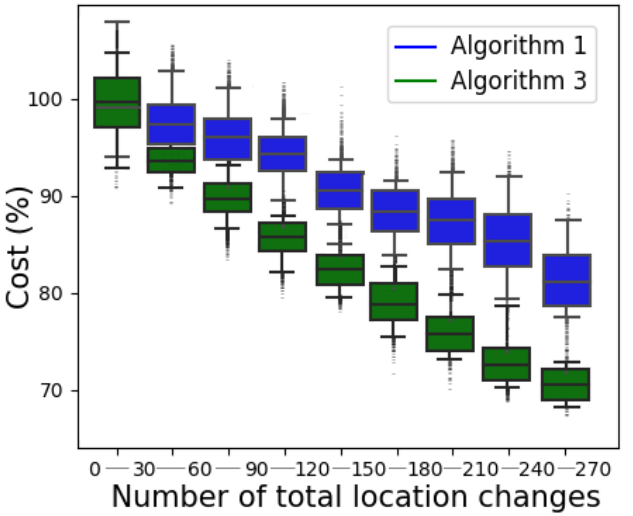

Fig. 3: The total number of product location reassignments needs to be large to achieve the best total travel costs in $f^*(x_{best})$ (dataset 2).

fit) between approximations f against f^*. For OR-tools, $R^2 = 0.97$ and for the heatmap, $R^2 = 0.15$. Even though the heatmap approximation is thousands of times faster to compute compared to TSP-optimizing the picking-log using OR-tools, OR-tools still results in more savings than the heatmap approximation. Due to its high speed, the heatmap approximation allows for more samples to be generated and higher initial savings, but due to its weaker predictive quality it, in the end, loses out to the TSP approximation.

The weakness of the heatmap approximation can be attributed to a combination of two factors. The first is that a swap of two products may result in a frequently picked product being located further from the depot, incurring an increased heatmap cost, while TSP distance, on the contrary, is reduced (this can be observed in Figure 8). The second factor is its bias to promote samples where high-frequency products are moved closer to depot. This type of bias risks leading the search to a pre-mature convergence on a local minimum. In order to prevent convergence on a local minimum, many samples are needed which temporarily increase TSP costs, but these types of samples are not often promoted in Algorithm 3 when the heatmap approximation is used.

Although OR-tools outperforms the heatmap approximation, one noted issue with the former is its high minimal CPU-time. The CPU-times of OR-tools are averaging 0.1 seconds to optimize a single TSP, whereas the corresponding CPU-time for Concorde is averaging 0.2 seconds. We could not achieve a lower value using the *solution_limit*

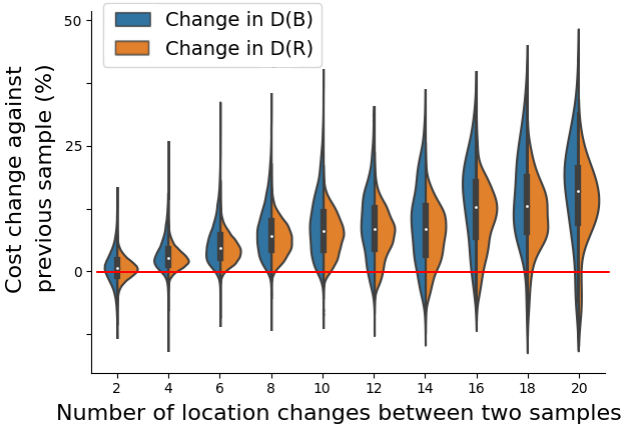

Fig. 4: Distribution (violin) plot showing number of location changes against picking-log distance $D(\mathcal{B})$ (blue) and reassignment distance $D(\mathcal{R})$ (orange) when moving from a previous sample to a new sample in the Markov Chain. The mean cost of both $D(\mathcal{B})$ and $D(\mathcal{R})$ increase when more location changes are attempted in new samples. This plot excludes any x_i and x_{i+1} pairs where either were restarts back to a local minimum.

parameter after several tests. On dataset 1 and 2, this CPU-time is potentially advantageous, since OR-tools delivers TSP distances within 1-2% of optimality (Table 2). This high approximation quality is explainable since pick-rounds $b \in B$ rarely exceed 15 locations in length in those datasets. On dataset 3, when the pick-rounds are 29 products on average, OR-tools is within 6% of optimality. We did not attempt to tune the CPU-time and the *solution_limit* parameter in OR-tools to maximize its utility within Algorithm 3.

Finally, we compute goodness of fit between both cost approximators and $R(D)$ for any generated samples (while λ was set to zero for dataset 3, $R(D)$ was still computed and logged). In both cases, R^2 was close to zero. While this may seem disadvantageous, it also means that $\mathcal{R}(\mathcal{D})$ has a high variance and low bias, thus preventing Algorithm 3 from converging on weak local minima. We also note that R^2 increases for promoted samples and even more so for accepted samples (reaching as high as $R^2 = 0.57$ for accepted samples). This provides further validation for Algorithm 3 and its cost function (Equation 1): The Markov Chain tends to converge on regions where picking-log cost is low and where reassignment costs are low as well.

Algorithm 1 compared to Algorithm 3 When the best settings found are utilized in Algorithm 3 (Nested Annealing with the OR-tools TSP cost approximation and restarts), it outperforms Algorithm 1 (Simulated Annealing without cost approximation and with restarts) within the given CPU-time (Figure 6). The Markov chain in

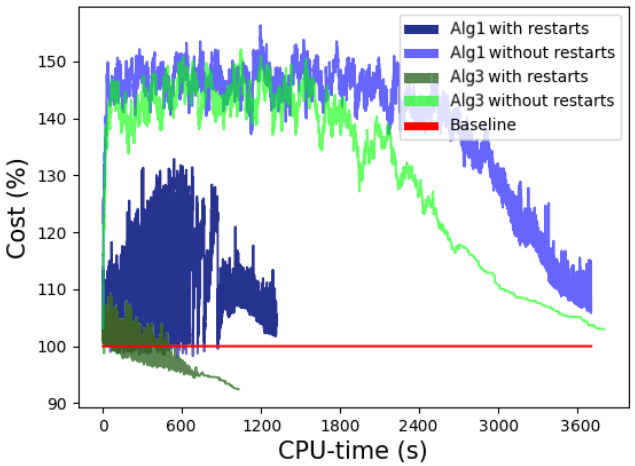

Fig. 5: Algorithm 1 and Algorithm 3 with and without *restarts* for 30000 iterations on dataset 2 [28]. The costs shown are for $f^*(x_{i+1})$.

Algorithm 3 is more biased compared to the one in Algorithm 1, due to more samples being rejected. Algorithm 1 searches through less attractive search regions, which reduces risk of convergence on local minima, so if given more CPU-time it could reach stronger results.

Other features Aggregate averages of results on the generated instances (dataset 1) and Algorithm 3 are shown in Table 1 (Appendix). The elements for columns $f(x_i)$, $f^*(x_i)$, $f(x_{i+1})$, $f^*(x_{i+1})$, $f^*(x_{best})$, $D(R)_1$ $D(R)_{300}$ are all shown as percentages against the distance of the baseline cost $f^*(x_1)$ (100%). $D(R)_1$ and $D(R)_{300}$ denote the distance of the reassignment path after Algorithm 2 has been run for 1 and 300 iterations, respectively. The rows are aggregated averages based on number of products shown in column 1, from a total of 5279885 samples on the instance set (with 3-12 minutes CPU-time on each instance).

The relationship between number of location changes and $D(\mathcal{R})$ can be seen in Figure 7. As more location swaps are carried out, the amount of reassignment distance increases, but the rate of increase slows down. One possible misconception is that the gradient should go down to zero as the reassignment path cannot exceed some hypothetical maximum. This is unlikely to occur, however, since the reassignment path may need to go back and forth through the warehouse several times to perform many reassignments.

No correlation was found between the warehouse layout and features such as total cost improvement, reassignment distance and/or number of final proposed location re-

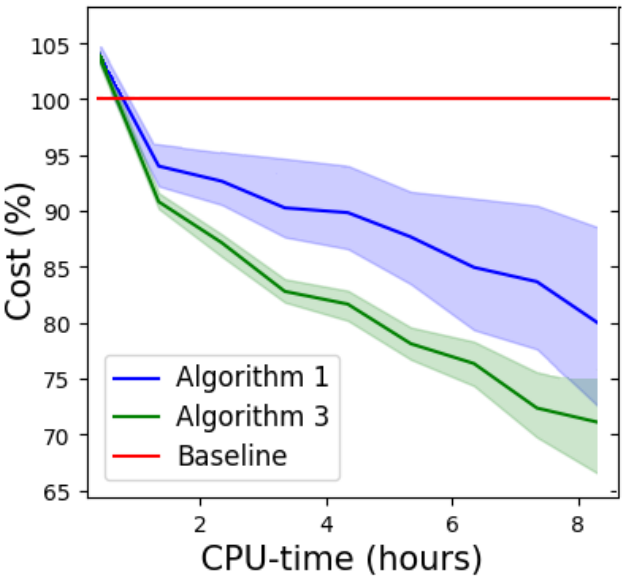

Fig. 6: Aggregate CPU-time against shortest total travel cost ($f^*(x_{best})$) on the real warehouse dataset (20 optimization runs): Blue is Algorithm 1, green is Algorithm 3 and red is the cost of baseline assignment x_1 (100%). The shadowed areas represent 95% confidence intervals [28]

.

assignments. This is explainable since both TSP-optimizers (OR-tools and Concorde) and the reassignment path optimizer (Algorithm 2) are *layout-agnostic* (Section 1).

7 Conclusion

This paper proposes a new optimization model for the Storage Assignment Location Problem (SLAP). In the Traveling Salesman Problem (TSP)-based SLAP, future forecasted picking is assumed to be static, while the warehouse rack layout can have any shape in two dimensions. In order to optimize the TSP-based SLAP, we propose a Nested Annealing algorithm. The algorithm is an extension of Simulated Annealing and generates assignment samples using a Hamming distance function and two sample filters. The algorithm requires fast and reasonable accurate cost approximations, and we propose two alternatives: One based on sub-optimal TSP optimization, and the other based on a pick-frequency heatmap. In order to reduce risk of convergence on a weak local minimum, we propose a *restart* heuristic, which forces occasional revisits to previously generated and relatively strong samples. Since products cannot be reassigned to new locations for free, a model for the *reassignment path* and *reassignment*

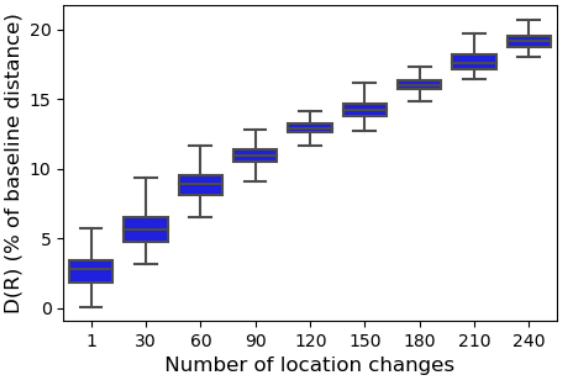

Fig. 7: Number of location changes vs. reassignment distance (as a percentage of baseline costs) (Algorithm 3 and dataset 3).

distance is proposed. This cost is computed and added to the cost of any generated sample.

To evaluate the proposed optimizer using various SLAP scenarios and optimization settings, experiments were conducted on three datasets: A set of publicly shared test-instances on the generalizable TSBLIB format, as well as two datasets from real warehouses. Results show that Nested Annealing yields cost savings of up to 30% within 8 hours of CPU-time. This result is in line with results in prior work, where strong assumptions are made with regard to warehouse layout (but where dynamicity may be included or where number of products is larger) [14, 17, 35]. Concerning the cost approximators, results show that sub-optimal TSP optimization outperforms the pick-frequency heatmap approach. While the former is thousands of times slower than the latter, it nevertheless achieves a better result due to its higher predictive accuracy.

For future work, heuristics to increase the amount of bias could be investigated. One cause of high variance in the proposed algorithm is that any product is allowed to swap location with any other product. Instead, products could be set up to be allocated to certain areas in the warehouse. This type of *zoning* is not trivially achieved, however, and could, if not carefully handled, lead to premature convergence on local minima. We concluded this after early tests and instead pursued cost approximation and the *promote* filter as another means to constrain the search space.

A topic which we did not explore extensively in this paper is the λ constant (Section 4) and its effect on optimization. We set it to either 0 or 1 and only concluded that it significantly slows down effective search when used. Instead of a constant, it could be set to change during optimization to potentially improve performance. For example, λ could be set to start at a low value and then grow linearly.

A final proposal involves analysis of the picking log and how it relates to potential cost savings. Zhang et al. [43] and Kofler et al. [14] use datamining heuristics to show that potential cost savings (the "reassignment potential") are correlated to the way in which products in pick-rounds are distributed. It is challenging to make use of such heuristics to make concrete proposals for reassignments in a Markov Chain, however. The SLAP remains a highly intractable problem.

Acknowledgements

This work was partially supported by the Wallenberg AI, Autonomous Systems and Software Program (WASP) funded by the Knut and Alice Wallenberg Foundation. We also convey thanks to Kairos Logic AB for software.

References

1. Applegate, D., Cook, W., Dash, S., Rohe, A.: Solution of a Min-Max Vehicle Routing Problem. INFORMS Journal on Computing **14**, 132–143 (2002)
2. Azadeh, K., De Koster, R., Roy, D.: Robotized and Automated Warehouse Systems: Review and Recent Developments. Transportation Science **53** (2019)
3. Boysen, N., Stephan, K.: The deterministic product location problem under a pick-by-order policy. Discrete Applied Mathematics **161**(18), 2862 – 2875 (2013)
4. Cardona, L.F., Rivera, L., Martínez, H.J.: Analytical study of the Fishbone Warehouse layout. International Journal of Logistics Research and Applications **15**(6), 365–388 (2012)
5. Charris, E., et al.: The storage location assignment problem: A literature review. International Journal of Industrial Engineering Computations **10** (2018)
6. Christen, J.A., Fox, C.: Markov Chain Monte Carlo Using an Approximation. Journal of Computational and Graphical Statistics **14**(4), 795–810 (2005), http://www.jstor.org/stable/27594150
7. Ene, S., Öztürk, N.: Storage location assignment and order picking optimization in the automotive industry. The International Journal of Advanced Manufacturing Technology **60**, 1–11 (May 2011). https://doi.org/10.1007/s00170-011-3593-y
8. Fontana, M.E., Nepomuceno, V.S.: Multi-criteria approach for products classification and their storage location assignment. The International Journal of Advanced Manufacturing Technology **88**(9), 3205–3216 (Feb 2017)
9. Garfinkel, M.: Minimizing multi-zone orders in the correlated storage assingment problem. PhD Thesis, School of Industrial and Systems Engineering, Georgia Institute of Technology (2005)
10. Gidas, B.: Nonstationary Markov chains and convergence of the annealing algorithm. Journal of Statistical Physics **39**(1), 73–131 (Apr 1985). https://doi.org/10.1007/BF01007975, https://doi.org/10.1007/BF01007975
11. Hahsler, M., Kurt, H.: TSP – Infrastructure for the Traveling Salesperson Problem. Journal of Statistical Software **2**, 1–21 (2007)
12. Henn, S., Wäscher, G.: Tabu search heuristics for the order batching problem in manual order picking systems. European Journal of Operational Research **222**(3), 484–494 (2012), publisher: Elsevier

13. Kallina, C., Lynn, J.: Application of the Cube-per-Order Index Rule for Stock Location in a Distribution Warehouse. Interfaces **7**(1), 37–46 (1976), http://www.jstor.org/stable/25059400
14. Kofler, M., Beham, A., Wagner, S., Affenzeller, M.: Affinity Based Slotting in Warehouses with Dynamic Order Patterns (Advanced Methods and Applications in Computational Intelligence), 123–143 (2014)
15. Koster, R.d., Le-Duc, T., Roodbergen, K.J.: Design and control of warehouse order picking: A literature review. European Journal of Operational Research **182**(2), 481 – 501 (2007)
16. Kruk, S.: Practical Python AI Projects: Mathematical Models of Optimization Problems with Google OR-Tools. Apress (2018)
17. Kübler, P., Glock, C., Bauernhansl, T.: A new iterative method for solving the joint dynamic storage location assignment, order batching and picker routing problem in manual picker-to-parts warehouses **147**, 106645 (2020)
18. Larco, J.A., Koster, R.d., Roodbergen, K.J., Dul, J.: Managing warehouse efficiency and worker discomfort through enhanced storage assignment decisions. International Journal of Production Research **55**(21), 6407–6422 (2017). https://doi.org/10.1080/00207543.2016.1165880, https://doi.org/10.1080/00207543.2016.1165880
19. Lee, I.G., Chung, S.H., Yoon, S.W.: Two-stage storage assignment to minimize travel time and congestion for warehouse order picking operations. Computers & Industrial Engineering **139**, 106129 (2020). https://doi.org/https://doi.org/10.1016/j.cie.2019.106129, https://www.sciencedirect.com/science/article/pii/S0360835219305984
20. Li, J., Moghaddam, M., Nof, S.Y.: Dynamic storage assignment with product affinity and ABC classification—a case study. The International Journal of Advanced Manufacturing Technology **84**(9), 2179–2194 (Jun 2016). https://doi.org/10.1007/s00170-015-7806-7, https://doi.org/10.1007/s00170-015-7806-7
21. Liu, C.M.: Clustering techniques for stock location and order-picking in a distribution center. Computers & Operations Research **26**(10), 989–1002 (1999). https://doi.org/https://doi.org/10.1016/S0305-0548(99)00026-X, https://www.sciencedirect.com/science/article/pii/S030505489900026X
22. Mackay, D.J.C.: Introduction to Monte Carlo Methods. In: Learning in Graphical Models (1998)
23. Mantel, R., et al.: Order oriented slotting: A new assignment strategy for warehouses. European Journal of Industrial Engineering **1**, 301–316 (2007)
24. Maruyama, K., Yamazaki, T.: Improved efficiency of warehouse picking by co-optimization of order batching and storage location assignment. Journal of Advanced Mechanical Design, Systems, and Manufacturing **16**(5), JAMDSM0052–JAMDSM0052 (2022). https://doi.org/10.1299/jamdsm.2022jamdsm0052
25. Ming-Huang Chiang, D., Lin, C.P., Chen, M.C.: Data mining based storage assignment heuristics for travel distance reduction. Expert Systems **31**(1), 81–90 (2014), publisher: Wiley Online Library
26. Oxenstierna, J., Krueger, V., Malec, J.: New benchmarks and optimization model for the Storage Location Assignment Problem. In: 3rd International Conference on Innovative Intelligent Industrial Production and Logistics, IN4PL 2022. SciTePress (2022)
27. Oxenstierna, J., Malec, J., Krueger, V.: Analysis of Computational Efficiency in Iterative Order Batching Optimization. In: Proceedings of the 11th International Conference on Operations Research and Enterprise Systems - ICORES,. pp. 345–353. SciTePress (2022). https://doi.org/10.5220/0010837700003117

28. Oxenstierna, J., Rensburg, L.v., Stuckey, P., Krueger, V.: Storage Assignment Using Nested Annealing and Hamming Distances. In: Proceedings of the 12th International Conference on Operations Research and Enterprise Systems - ICORES,. pp. 94–105. SciTePress (2023). https://doi.org/10.5220/0011785100003396, backup Publisher: INSTICC ISSN: 2184-4372
29. Rajasekaran, S., Reif, J.H.: Nested annealing: a provable improvement to simulated annealing. Theoretical Computer Science **99**(1), 157–176 (1992)
30. Rathod, A.B., Gulhane, S.M., Padalwar, S.R.: A comparative study on distance measuring approches for permutation representations. In: 2016 IEEE international conference on advances in electronics, communication and computer technology (ICAECCT). pp. 251–255. IEEE (2016)
31. Janse van Rensburg, L.J.v.: Artificial intelligence for warehouse picking optimization - an NP-hard problem. Master's thesis, Uppsala University (2019)
32. Roodbergen, K.J., Koster, R.: Routing methods for warehouses with multiple cross aisles. International Journal of Production Research **39**(9), 1865–1883 (2001)
33. Schapire, R.: Using Output Codes to Boost Multiclass Learning Problems (Feb 2001)
34. Tak, H., Meng, X.L., Dyk, D.A.v.: A Repelling–Attracting Metropolis Algorithm for Multimodality. Journal of Computational and Graphical Statistics **27**(3), 479–490 (Jul 2018). https://doi.org/10.1080/10618600.2017.1415911,
35. Trindade, M.A.M., Sousa, P., Moreira, M.: Ramping up a heuristic procedure for storage location assignment problem with precedence constraints. Flexible Services and Manufacturing Journal **34** (09 2022). https://doi.org/10.1007/s10696-021-09423-w
36. Valle, C., Beasley, J.E., da Cunha, A.S.: Optimally solving the joint order batching and picker routing problem. European Journal of Operational Research **262**(3), 817–834 (2017)
37. Wales, D.J., Doye, J.P.K.: Global Optimization by Basin-Hopping and the Lowest Energy Structures of Lennard-Jones Clusters Containing up to 110 Atoms. Journal of Physical Chemistry A **101**, 5111–5116 (1997)
38. Wu, J., Qin, T., Chen, J., Si, H., Lin, K.: Slotting Optimization Algorithm of the Stereo Warehouse. In: Proceedings of the 2012 2nd International Conference on Computer and Information Application (ICCIA 2012). pp. 128–132. Atlantis Press (May 2014). https://doi.org/https://doi.org/10.2991/iccia.2012.31, https://doi.org/10.2991/iccia.2012.31
39. Wutthisirisart, P., Noble, J.S., Chang, C.A.: A two-phased heuristic for relation-based item location. Computers & Industrial Engineering **82**, 94–102 (2015). https://doi.org/https://doi.org/10.1016/j.cie.2015.01.020, https://www.sciencedirect.com/science/article/pii/S036083521500039X
40. Xiang, X., Liu, C., Miao, L.: Storage assignment and order batching problem in Kiva mobile fulfilment system. Engineering Optimization **50**(11), 1941–1962 (2018). https://doi.org/10.1080/0305215X.2017.1419346, https://doi.org/10.1080/0305215X.2017.1419346
41. Yu, M., Koster, R.B.M.d.: The impact of order batching and picking area zoning on order picking system performance. European Journal of Operational Research **198**(2), 480 – 490 (2009)
42. Yu, V.F., Winarno, Maulidin, A., Redi, A.A.N.P., Lin, S.W., Yang, C.L.: Simulated Annealing with Restart Strategy for the Path Cover Problem with Time Windows. Mathematics **9**(14) (2021). https://doi.org/10.3390/math9141625, https://www.mdpi.com/2227-7390/9/14/1625
43. Zhang, R.Q., et al.: New model of the storage location assignment problem considering demand correlation pattern. Computers & Industrial Engineer-

ing **129**, 210–219 (2019). https://doi.org/https://doi.org/10.1016/j.cie.2019.01.027, https://www.sciencedirect.com/science/article/pii/S0360835219300294
44. Žulj, I., Glock, C.H., Grosse, E.H., Schneider, M.: Picker routing and storage-assignment strategies for precedence-constrained order picking. Computers & Industrial Engineering **123**, 338–347 (2018). https://doi.org/https://doi.org/10.1016/j.cie.2018.06.015, https://www.sciencedirect.com/science/article/pii/S0360835218302869

8 Appendix

Examples of pick-rounds before and after 100 iterations of SLAP optimization (left and right respectively). The SLAP can be challenging even when there are only six pick-rounds in the picking-log. While it is relatively easy to spot suitable swaps of locations for pick-rounds involving few products, it is more difficult when pick-rounds are long. One of the products is picked in all of the pick-rounds, and as that product is moved, it affects total distance in an unforseeable manner.

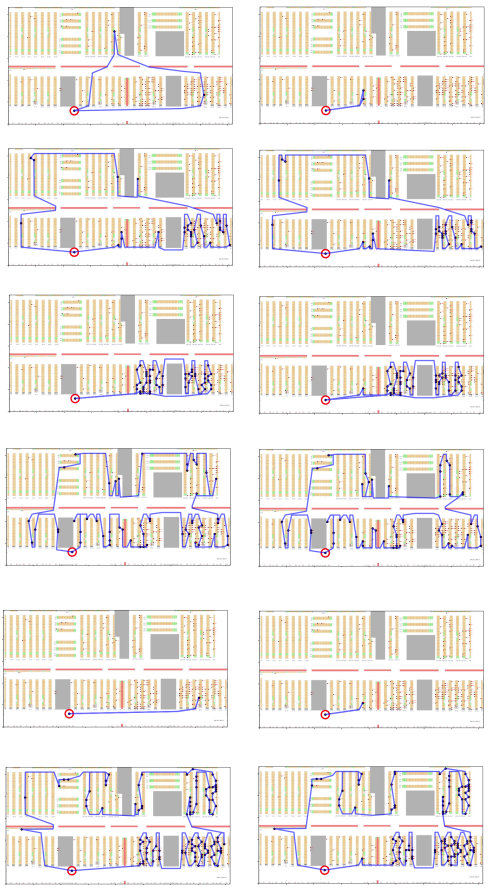

Fig. 8: Pictures of optimally solved pick-rounds (TSP's) before (left) and after SLAP optimization (right). The product which is picked in all pick-rounds is the lower-rightmost one in the upper two pictures (before and after it was moved).

Table 1: Aggregate averages of results from 5279885 generated samples for optimization runs on the 266 publicly shared instances. The results are aggregated based on ranges of number of products (the first column).

Num_products	Num_pick-rounds	Num_samples	CPU-time (s)	$f^*(x_i)$	$f(x_i)$	$f^*(x_{i+1})$	$f(x_{i+1})$	$D(R)_1$	$f^*(x_{best})$	$D(R)_{300}$	Number of location changes
5	7	39997	215	89	89	109	98	85.3	27.62	24.83	7.3
14	9	319966	254	89	89	110	99	88.0	26.80	24.31	8.5
23	11	439980	293	91	91	106	101	88.4	24.48	22.35	12.9
32	14	439983	281	94	93	107	102	91.8	21.06	19.27	12.9
41	19	459987	269	96	96	107	103	93.1	17.35	15.85	13.2
50	23	519994	295	96	96	104	102	93.5	15.89	14.58	15.2
60	22	459991	344	97	97	105	103	94.5	15.33	13.94	14.3
69	26	339997	313	97	97	104	103	94.8	13.37	12.17	14.2
78	25	279995	335	97	97	105	103	94.1	13.44	12.23	15.1
87	28	380000	372	98	98	104	103	96.3	12.08	10.95	14.0
96	33	200000	373	98	98	104	102	94.9	12.31	11.20	15.2
106	24	119999	410	97	97	103	102	95.4	12.22	10.99	12.9
115	29	179998	462	99	99	105	103	97.4	10.43	9.40	12.5
124	32	180000	487	99	99	104	103	97.6	9.83	8.78	11.4
133	32	120000	455	99	99	104	103	97.1	9.27	8.40	12.5
142	36	60000	449	98	98	103	102	97.5	9.87	8.95	11.8
151	35	79999	527	98	98	103	102	97.4	11.21	10.11	15.8
161	39	60000	476	99	99	103	102	98.6	7.48	6.74	12.3
170	38	119999	551	99	99	103	102	97.7	9.30	8.40	14.5
179	37	80000	576	100	100	104	103	98.3	7.24	6.50	10.7
188	37	60000	583	100	100	104	103	99.7	7.21	6.49	11.3
197	32	20000	693	100	100	104	103	99.7	8.88	7.93	11.0
216	44	20000	552	101	100	104	103	99.8	5.68	5.03	8.3
225	38	40000	655	100	100	104	103	98.7	7.99	7.17	11.9
234	36	20000	682	99	98	103	102	97.7	8.11	7.31	13.8
252	39	20000	825	100	100	104	103	99.7	6.36	5.70	10.2
262	44	20000	624	100	99	103	102	99.0	6.66	5.95	12.0
289	40	40000	821	100	100	104	103	99.4	6.38	5.68	11.5
317	161	20000	629	100	100	101	101	99.2	4.35	3.95	17.9
326	162	20000	636	99	99	101	101	99.2	3.66	3.32	15.5
381	206	20000	723	100	100	101	101	99.3	2.89	2.62	18.2
390	207	20000	719	100	100	101	101	99.6	2.78	2.50	13.9
399	205	20000	735	100	100	101	101	99.6	3.11	2.81	16.3
418	211	40000	757	100	100	101	101	99.7	2.73	2.45	13.1
427	222	20000	806	100	100	101	101	99.9	2.41	2.15	13.7

Table 2: Aggregate averages of results on dataset 3, where the two cost approximators are compared. The CPU-times are here for predictions of single TSPs.

CPU-time total (s)	Heatmap					OR-tools					CPU-time Concorde (s)
	CPU-time (s)	$f(x_t)$	$f^*(x_t)$	$f(x_{t+1})$	$f^*(x_{t+1})$	CPU-time (s)	$f(x_t)$	$f^*(x_t)$	$f(x_{t+1})$	$f^*(x_{t+1})$	
0	3.7E-05	91.51	106.63	96.27	112.34	0.098	107.33	101.58	110.58	104.33	0.20
270	3.5E-05	103.32	98.90	111.67	106.26	0.096	105.97	98.70	112.11	103.82	0.26
543	3.6E-05	97.47	96.78	104.30	102.54	0.096	103.85	95.65	110.98	102.11	0.24
816	3.8E-05	88.73	96.29	95.32	103.28	0.102	101.93	95.01	107.07	99.13	0.15
1089	3.7E-05	93.56	92.87	100.61	98.94	0.099	102.67	95.13	106.29	98.45	0.28
1362	3.9E-05	100.55	95.32	107.56	100.74	0.101	101.87	93.99	104.47	96.32	0.37
1635	3.7E-05	98.19	89.44	100.68	93.41	0.095	101.53	93.16	105.50	97.02	0.24
1908	3.7E-05	92.69	94.06	90.60	99.75	0.101	99.82	92.16	104.77	96.55	0.12
2181	3.9E-05	97.68	93.58	95.73	97.06	0.098	99.32	91.68	103.81	95.37	0.23
2454	3.9E-05	92.99	93.09	90.73	97.05	0.098	99.44	91.90	102.96	95.09	0.33
2727	4.2E-05	95.97	91.94	93.89	97.27	0.099	98.29	90.27	102.37	93.51	0.30
3000	4.2E-05	99.26	97.54	97.64	103.29	0.104	97.28	87.64	100.91	92.80	0.32
3273	3.9E-05	98.11	99.43	97.15	105.25	0.101	93.52	87.75	99.65	92.12	0.27
3546	3.7E-05	103.19	96.21	102.50	101.50	0.104	93.65	87.56	97.72	92.41	0.19
3819	3.8E-05	101.53	90.65	99.49	96.67	0.103	93.47	87.05	98.73	92.77	0.22
4092	3.7E-05	97.16	82.51	95.38	87.16	0.106	93.21	86.11	97.84	91.10	0.26
4365	3.5E-05	89.30	89.68	87.11	93.44	0.100	92.52	83.55	96.28	89.34	0.16
4638	3.6E-05	96.11	86.36	94.77	91.63	0.105	92.44	83.72	96.29	89.56	0.34
4911	3.5E-05	87.10	90.53	85.77	96.55	0.102	90.74	83.19	94.42	89.14	0.15
5184	3.5E-05	95.06	80.34	93.42	84.97	0.101	89.85	85.54	94.84	88.84	0.26
5457	4.2E-05	92.76	84.32	92.28	88.70	0.105	90.21	85.04	94.44	88.84	0.21
5730	3.9E-05	89.06	84.80	91.24	89.46	0.100	90.47	85.48	94.24	88.71	0.29
6003	3.9E-05	92.65	86.15	93.86	92.10	0.102	89.95	84.55	93.92	87.65	0.22
6276	3.9E-05	99.31	85.65	99.20	91.94	0.100	88.25	83.61	91.95	86.30	0.18
6549	3.8E-05	100.67	78.19	100.20	85.76	0.102	87.33	82.98	91.57	85.45	0.18
6822	3.7E-05	99.41	79.06	100.38	83.74	0.105	84.69	82.24	88.97	84.57	0.10
7095	3.8E-05	89.81	77.54	89.49	83.44	0.105	84.93	81.31	89.00	83.14	0.19
7368	4.1E-05	97.47	83.69	95.78	88.54	0.100	83.83	78.13	88.31	83.35	0.21
7641	3.8E-05	82.96	83.08	78.77	88.70	0.103	83.15	78.36	87.61	83.59	0.22
7914	3.6E-05	94.68	86.11	90.21	90.42	0.098	83.51	78.08	86.89	82.65	0.25
8187	3.7E-05	99.96	85.77	95.34	91.37	0.101	82.51	75.72	86.83	81.83	0.16
8460	3.6E-05	96.36	82.89	91.15	88.68	0.099	81.45	75.62	85.25	80.90	0.11
8733	3.7E-05	90.84	81.15	85.80	88.06	0.099	82.36	74.71	85.72	80.28	0.08
9006	3.7E-05	100.12	79.78	95.60	85.21	0.098	80.37	74.88	83.76	79.62	0.21
9279	3.5E-05	98.74	80.18	93.87	85.31	0.101	80.55	74.82	83.82	78.79	0.21
9552	3.6E-05	89.82	79.70	85.77	84.68	0.105	79.34	74.82	82.63	72.64	0.27
9825	3.5E-05	91.59	77.13	87.20	81.96	0.098	79.89	73.46	82.04	72.33	0.15
10098	3.4E-05	95.74	83.77	93.14	87.98	0.099	80.07	73.34	82.27	72.58	0.15
10371	3.3E-05	93.30	78.10	93.16	83.05	0.097	80.03	75.46	82.61	72.64	0.08
10644	3.3E-05	87.79	77.62	86.76	82.77	0.099	79.11	72.41	81.55	72.11	0.18
10917	3.3E-05	83.07	88.14	82.73	81.81	0.101	78.38	71.94	80.03	71.55	0.12
11190	3.3E-05	91.08	86.66	90.38	81.21	0.105	77.78	71.47	79.69	71.82	0.12
11463	3.2E-05	83.91	86.18	84.08	80.54	0.100	77.02	71.35	79.19	71.97	0.24
11736	3.3E-05	81.51	82.23	80.71	86.52	0.107	77.88	71.12	80.14	71.40	0.18
12009	3.3E-05	80.99	84.06	80.05	88.76	0.104	76.13	73.06	78.06	72.34	0.16
12282	3.3E-05	90.28	85.41	89.69	80.27	0.102	74.89	71.93	77.25	72.07	0.26
12555	3.3E-05	77.18	84.25	76.41	80.76	0.106	74.14	71.24	77.00	71.76	0.21
12828	3.4E-05	71.68	88.43	71.20	85.31	0.103	73.66	71.03	76.55	71.13	0.31
13101	3.4E-05	69.98	83.28	69.62	88.12	0.102	73.12	70.92	76.07	71.07	0.20