Optimization of storage and picking systems in warehouses

Thèse

Allyson Fernandes da Costa Silva

Doctorat en Sciences de l'administration – opérations et systèmes de décision
Philosophiæ doctor (Ph. D.)

Québec, Canada

Optimization of storage and picking systems in warehouses

Thèse

Allyson Fernandes da Costa Silva

Sous la direction de:

Prof. Leandro C. Coelho, directeur de recherche
Prof. Maryam Darvish, codirectrice de recherche

Résumé

La croissance du commerce électronique exige une hausse des performances des systèmes d'entreposage, qui sont maintenant repensés pour faire face à un volume massif de demandes à être satisfait le plus rapidement possible. Le système manuel et le système à robots mobile (SRM) sont parmi les plus utilisés pour ces activités. Le premier est un système centré sur l'humain pour réaliser des opérations complexes que les robots actuels ne peuvent pas effectuer. Cependant, les nouvelles générations de robots autonomes mènent à un remplacement progressif par le dernier pour augmenter la productivité.

Quel que soit le système utilisé, plusieurs problèmes interdépendants doivent être résolus pour avoir des processus de stockage et de prélèvement efficaces. Les problèmes de stockage concernent les décisions d'où stocker les produits dans l'entrepôt. Les problèmes de prélèvement incluent le regroupement des commandes à exécuter ensemble et les itinéraires que les cueilleurs et les robots doivent suivre pour récupérer les produits demandés. Dans le système manuel, ces problèmes sont traditionnellement résolus à l'aide de politiques simples que les préparateurs peuvent facilement suivre. Malgré l'utilisation de robots, la même stratégie de solution est répliquée aux problèmes équivalents trouvés dans le SRM.

Dans cette recherche, nous étudions les problèmes de stockage et de prélèvement rencontrés lors de la conception du système manuel et du SRM. Nous développons des outils d'optimisation pour aider à la prise de décision pour mettre en place leurs processus, en améliorant les mesures de performance typiques de ces systèmes. Certains problèmes traditionnels sont résolus avec des techniques améliorées, tandis que d'autres sont intégrés pour être résolus ensemble au lieu d'optimiser chaque sous-système de manière indépendante.

Nous considérons d'abord un système manuel avec un ensemble connu de commandes et intégrons les décisions de stockage et de routage. Le problème intégré et certaines variantes tenant compte des politiques de routage communes sont modélisés mathématiquement. Une métaheuristique générale de recherche de voisinage variable est présentée pour traiter des instances de taille réelle. Des expériences attestent de l'efficience de la métaheuristique proposée par rapport aux modèles exacts et aux politiques de stockage communes.

Lorsque les demandes futures sont incertaines, il est courant d'utiliser une stratégie de zonage

qui divise la zone de stockage en zones et attribue les produits les plus demandés aux meilleures zones. Les tailles des zones sont à déterminer. Généralement, des dimensions arbitraires sont choisies, mais elles ignorent les caractéristiques de l'entrepôt et des demandes. Nous abordons le problème de dimensionnement des zones pour déterminer quels facteurs sont pertinents pour choisir de meilleures tailles de zone. Les données générées à partir de simulations exhaustives sont utilisées pour trainer quatre modèles de régression d'apprentissage automatique – moindres carrés ordinaire, arbre de régression, forêt aléatoire et perceptron multicouche – afin de prédire les dimensions optimales des zones en fonction de l'ensemble de facteurs pertinents identifiés. Nous montrons que tous les modèles entraînés suggèrent des dimensions sur mesure des zones qui performent meilleur que les dimensions arbitraires couramment utilisées.

Une autre approche pour résoudre les problèmes de stockage pour le système manuel et pour le SRM considère les corrélations entre les produits. L'idée est que les produits régulièrement demandés ensemble doivent être stockés près pour réduire les coûts de routage. Cette politique de stockage peut être modélisée comme une variante du problème d'affectation quadratique (PAQ). Le PAQ est un problème combinatoire traditionnel et l'un des plus difficiles à résoudre. Nous examinons les variantes les plus connues du PAQ et développons une puissante métaheuristique itérative de recherche tabou mémétique en parallèle capable de les résoudre. La métaheuristique proposée s'avère être parmi les plus performantes pour le PAQ et surpasse considérablement l'état de l'art pour ses variantes.

Les SRM permettent de repositionner facilement les pods d'inventaire pendant les opérations, ce qui peut conduire à un processus de prélèvement plus économe en énergie. Nous intégrons les décisions de repositionnement des pods à l'attribution des commandes et à la sélection des pods à l'aide d'une stratégie de prélèvement par vague. Les pods sont réorganisés en tenant compte du moment et de l'endroit où ils devraient être demandés au futur. Nous résolvons ce problème en utilisant la programmation stochastique en tenant compte de l'incertitude sur les demandes futures et suggérons une matheuristique de recherche locale pour résoudre des instances de taille réelle. Nous montrons que notre schéma d'approximation moyenne de l'échantillon est efficace pour simuler les demandes futures puisque nos méthodes améliorent les solutions trouvées lorsque les vagues sont planifiées sans tenir compte de l'avenir.

Cette thèse est structurée comme suit. Après un chapitre d'introduction, nous présentons une revue de la littérature sur le système manuel et le SRM, et les décisions communes prises pour mettre en place leurs processus de stockage et de prélèvement. Les quatre chapitres suivants détaillent les études pour le problème de stockage et de routage intégré, le problème de dimensionnement des zones, le PAQ et le problème de repositionnement de pod. Nos conclusions sont résumées dans le dernier chapitre.

Keywords : Conception d'entrepôt ; systèmes de stockage et de prélèvement ; heuristiques ; parallélisme ; apprentissage automatique ; programmation stochastique

Abstract

The rising of e-commerce is demanding an increase in the performance of warehousing systems, which are being redesigned to deal with a mass volume of demands to be fulfilled as fast as possible. The manual system and the robotic mobile fulfillment system (RMFS) are among the most commonly used for these activities. The former is a human-centered system that handles complex operations that current robots cannot perform. However, newer generations of autonomous robots are leading to a gradual replacement by the latter to increase productivity.

Regardless of the system used, several interdependent problems have to be solved to have efficient storage and picking processes. Storage problems concern decisions on where to store products within the warehouse. Picking problems include the batching of orders to be fulfilled together and the routes the pickers and robots should follow to retrieve the products demanded. In the manual system, these problems are traditionally solved using simple policies that pickers can easily follow. Despite using robots, the same solution strategy is being replicated to the equivalent problems found in the RMFS.

In this research, we investigate storage and picking problems faced when designing manual and RMFS warehouses. We develop optimization tools to help in the decision-making process to set up their processes and improve typical performance measures considered in these systems. Some classic problems are solved with improved techniques, while others are integrated to be solved together instead of optimizing each subsystem sequentially.

We first consider a manual system with a known set of orders and integrate storage and routing decisions. The integrated problem and some variants considering common routing policies are modeled mathematically. A general variable neighborhood search metaheuristic is presented to deal with real-size instances. Computational experiments attest to the effectiveness of the metaheuristic proposed compared to the exact models and common storage policies.

When future demands are uncertain, it is common to use a zoning strategy to divide the storage area into zones and assign the most-demanded products to the best zones. Zone sizes are to be determined. Commonly, arbitrary sizes are chosen, which ignore the characteristics of the warehouse and the demands. We approach the zone sizing problem to determine which

factors are relevant to choosing better zone sizes. Data generated from exhaustive simulations are used to train four machine learning regression models – ordinary least squares, regression tree, random forest, and multilayer perceptron – to predict the optimal zone sizes given the set of relevant factors identified. We show that all trained models suggest tailor-made zone sizes with better picking performance than the arbitrary ones commonly used.

Another approach to solving storage problems, both in the manual and RMFS, considers the correlations between products. The idea is that products constantly demanded together should be stored closer to reduce routing costs. This storage policy can be modeled as a quadratic assignment problem (QAP) variant. The QAP is a traditional combinatorial problem and one of the hardest to solve. We survey the most traditional QAP variants and develop a powerful parallel memetic iterated tabu search metaheuristic capable of solving them. The proposed metaheuristic is shown to be among the best performing ones for the QAP and significantly outperforms the state-of-the-art for its variants.

The RMFS allows easy repositioning of inventory pods during operations that can lead to a more energy-efficient picking process. We integrate pod repositioning decisions with order assignment and pod selection using a wave picking strategy such that pods are parked after being requested considering when and where they are expected to be requested next. We solve this integrated problem using stochastic programming considering the uncertainty about future demands and suggest a local search matheuristic to solve real-size instances. We show that our sample average approximation scheme is effective to simulate future demands since our methods improve solutions found when waves are planned without considering the future demands.

This thesis is structured as follows. After an introductory chapter, we present a literature review on the manual and RMFS, and common decisions made to set up their storage and picking processes. The next four chapters detail the studies for the integrated storage and routing problem, the zone sizing problem, the QAP, and the pod repositioning problem. Our findings are summarized in the last chapter.

Keywords: Warehouse design; storage and picking systems; heuristics; parallel computing; machine learning; stochastic programming

Contents

List of Tables

List of Figures

To my loved Raiana
and our dreams

Acknowledgements

I dedicate my first words to Professor Leandro Coelho from whom I learned that giving my best is always the least I can do. His exceptional dedication and commitment to work are inspiring. I also feel privileged to have Professor Maryam Darvish in the direction of my research. Many times in these four years I have wondered how fortunate I have been for having these two brilliant and supportive mentors. I thank them for keeping pushing.

I would like to thank all the professors whose direct contribution made this possible. Special thanks to Professor Daniel Aloise, whom I will forever be grateful for showing me the beauty of the operations research, and to Professors Jacques Renaud and Kees Jan Roodbergen, whose collaborations in the projects developed during these years were essential to delivering results with the highest quality. I also thank the jury members, Professors Adriana Gabor, Anand Subramanian, and Walter Rei, for the time dedicated to reading and assessing the quality of this thesis.

I am thankful for all the staff members at the *Faculté des sciences de l'administration* (FSA), the *Centre interuniversitaire de recherche sur les réseaux d'entreprise, la logistique et le transport* (CIRRELT), and the Compute Canada for providing the best support, tools, and environment to make this accomplishment possible.

I also acknowledge the financing of this research provided through the many scholarships and awards granted. For this reason, I thank the Canada Research Chair in Integrated Logistics, the FSA, the CIRRELT, the *Centre d'innovation en logistique et chaîne d'approvisionnement durable* (CILCAD), and the donors of the François Rouette-Cain Lamarre and the Gervaise Cimon scholarships.

I thank my beloved parents, Clezimar and Marcia, and my brothers, Anderson and Emerson. Despite the distance, they will forever be my foundation. I still thank all the old and new friends that contributed to making this dream of moving to a new place more pleasant. I will miss the regular escapes to Montreal to play board games.

Finally, I cannot express how thankful I am for my wife, Raiana Pinheiro, the most important person in my life. I am deepest grateful for her care and love.

Preface

This thesis presents my work as a Ph.D. student completed during the program of study at the *Faculté des sciences de l'administration* at the *Université Laval*. The thesis consists of five chapters, where three among them are written in collaboration with other researchers, mainly my director Prof. Leandro C. Coelho and co-director Prof. Maryam Darvish. Two of these papers are published and one is currently under revision. In all papers, I remain the first author and have played the major role of setting up and conducting the research, modeling the problem, implementing the algorithms, analyzing the results, and preparing and writing the papers.

The first paper entitled "Integrating storage location and order picking problems in warehouse planning" is written in collaboration with Leandro C. Coelho, Maryam Darvish, and Jacques Renaud. The paper was accepted in *Transportation Research Part E: Logistics and Transportation Review* on May 31, 2020 and published on August 21, 2020.

The second paper entitled "Quadratic assignment problem variants: a survey and an effective parallel memetic iterated tabu search" is written in collaboration with Leandro C. Coelho, and Maryam Darvish. The paper was accepted in *European Journal of Operational Research* on November 20, 2020 and published online on November 26, 2020.

The third paper entitled "Estimating optimal ABC zone sizes in manual warehouses" is written in collaboration with Kees Jan Roodbergen, Leandro C. Coelho, and Maryam Darvish.

Introduction

The efficiency of warehouse operations is vital for determining a company's competitiveness since logistic costs constitute an important part of the overall costs (Rouwenhorst et al., 2000). The importance of having an efficient warehouse management system has been increasing in the past years with the changes in the global markets. Businesses have moved to places where labor costs are lower as an effect of globalization. The increasing sales volumes of e-commerce, which skyrocketed after the restrictions imposed to control the spread of the Covid-19 virus, require new business strategies related to logistic operations. Also, the development of new information technologies has made goods move faster than before (Boysen et al., 2019a; Ho and Tseng, 2006; Yener and Yazgan, 2019).

Warehouse processes may be summarized to receiving, storage, picking, and shipping (Rouwenhorst et al., 2000). While the receiving and the shipping have their interesting problems, it is at the storage and picking that the biggest part of the warehouse operations is performed. Warehouses are often optimized for cost-efficient order picking considering resource constraints such as labor, capital, and space (De Koster et al., 2007; Van Gils et al., 2018c).

The storage system refers to the system used to store products in storage locations. It consists of multiple subsystems comprising the policy used to define storage locations, the storage and the rearrangement processes at the storage area, and the technologies used for the storage system. The picking system refers to the system used to pick products from storage locations. Its subsystems are related to the batch of orders to create the pick lists, the zoning of the warehouse, the routes followed by pickers to retrieve items, and all technologies and resources involved in these operations.

The storage and picking processes in a manual warehouse are performed by human pickers who walk through the storage area collecting the requested products at their storage locations (Petersen, 1997, 1999). Many automated systems in use eliminate the unproductive walking of pickers by bringing products to fixed picking stations located around the storage area (Azadeh et al., 2019). More recently, the robotic mobile fulfillment system (RMFS), where robots can lift inventory pods and bring them to stationary pickers, has been successfully adopted in e-commerce warehouses (Boysen et al., 2019a).

Designing an efficient storage and picking system involves several interrelated decisions. For this reason, the choice of methods to solve them is highly dependent on each other. No overall accepted systematic procedure exists to design such systems (Rouwenhorst et al., 2000). Nevertheless, some decisions to be made are common regardless of the system used.

The problem that involves the assignment of products to storage locations is known as *storage location assignment problem* (SLAP). Overall, products have to be placed in convenient locations to be easily picked during the picking process. Its objective is to optimize a performance measure associated with space and distance, normally the minimization of travel distances (Reyes et al., 2019). Occasionally, products' locations have to be rearranged due to factors that may affect product demands (Carlo and Giraldo, 2012). Rearrangements of products' locations should be done when savings on overall picking costs and effort are higher than the cost of the process to rearrange them. In manual warehouses, rearrangements are costly and, therefore, infrequent. However, pods can be easily moved around in an RMFS. The location where pods should park after a pick is a decision problem known as *pod repositioning problem* (PRP) (Xie et al., 2021).

The travel distances used to evaluate solutions for the SLAP and PRP are determined by the routes followed either by pickers or robots. This routing problem is known as the *order picking problem* (OPP) (Pansart et al., 2018; Scholz et al., 2016). In a manual system, routes contain the multiple storage locations visited to retrieve all products in the pick list. On the other hand, a robot in an RMFS follows a straightforward route from the pod location to the workstation where the pick is performed. However, since products are usually scattered among multiple pods, the pod to be selected is a decision problem known as *pod selection problem* (PSP) (Weidinger and Boysen, 2018).

Combining several orders into batches has been shown to reduce the total picking time significantly in manual systems (Petersen and Aase, 2004). The *order batching problem* (OBP) consists in determining which set of orders will be treated together by a single picker. In an RMFS, orders are usually assigned to stations using an online picking strategy, i.e., decisions are made in real-time. In this case, the OBP is modified to the *order assignment problem* (OAP) in which an order from the backlog is assigned to a workstation considering the current state of the system (Merschformann et al., 2019). Batching can be adopted in RMFS to reduce the number of visits of pods to stations whenever order assignment decisions can be done periodically (wave picking). This can have a significant impact on the robot utilization associated costs.

All these problems have already been studied in the literature (De Koster et al., 2007; Gu et al., 2007; Merschformann et al., 2019). Even as individual problems, most are considered to be hard to solve. The SLAP is related to the quadratic assignment problem (QAP), which is one of the hardest combinatorial problems to solve (Drezner, 2008). The OPP is a

special case of the traditional travelling salesman problem (Scholz et al., 2016). The OAP is usually modeled as a clustering problem (Li et al., 2020; Mirzaei et al., 2021). The OBP is formulated as a set partitioning problem (Gademann and Velde, 2005). The PRP and PSP require some knowledge about the stochastic nature of future demands to be solved efficiently (Merschformann et al., 2019). Many effort is being dedicated in the literature to integrate them (Dijkstra and Roodbergen, 2017; Van Gils et al., 2018c). However, integrating decisions further increases the complexity of the problem to be solved.

The recent advances in computational technologies (parallel computing) and solution techniques (hybrid metaheuristics, machine learning, stochastic optimization) enable these hard problems to be solved by complex models resulting from their efficient integration. In this research, we develop tools to help design efficient warehousing systems, both manual and RMFS. The problems previously described are modeled and solved either as individual problems or as integrated ones considering the current technological development. Exact methods are proposed to find optimal solutions for them. Due to their complexity, alternative heuristic approaches are also suggested to deal with real-size instances. The objective is to solve classical problems more efficiently than using the methods previously proposed, and to integrate them with others, whenever applicable.

The remainder of this thesis is organized as follows. In Chapter 1, we analyze the literature on the design of storage and picking systems, particularly the manual system and the RMFS. We present significant studies that investigate tactical and operational problems commonly found in the design of these two systems related to storage, batching, and routing. A description of common methods to solve them is provided, and part of the chapter is dedicated to introducing studies that investigated the interactions between them. Also, we provide a brief discussion of the literature analyzed and pinpoint some pertinent directions for future research.

In Chapter 2, we introduce and model the storage location and order picking problem for a manual warehouse, which consists of the integration of the SLAP and the OPP solved as a deterministic problem. The problem considers that a set of pick lists representing the future demands is known. Then, the storage area is organized such that the total traveling distance to be followed by the pickers to retrieve the products contained in these pick lists is minimized. The model is adapted to four special cases with imposed heuristic routing policies commonly found for the manual warehouse. The resulting non-linear models are linearized. Experiments show that even the linear models are difficult to be solved up to optimality. Therefore, we present a metaheuristic that is observed to be very efficient for real size problems. The results obtained significantly improve solutions generated by common storage policies, which are used to solve the SLAP as an individual problem.

In Chapter 3, the assumption that we have total knowledge of future demands is relaxed. We are now interested in assigning products to zones of different sizes to be determined. The goal

is to minimize the expected average route length. While popular, solving the SLAP with an ABC storage policy with arbitrary and standard sizes for each zone can lead to considerable efficiency loss in many common warehouse settings. In this chapter, we investigate the zone sizing problem. We implemented a simulator to estimate the average route length given factors related to the warehouse layout, the pick list characteristics, and the rules used to decide the shape of zones and to create the routes. An exhaustive search is done for several combinations of zone sizes. Those that result in the lowest route lengths are analyzed to evaluate which factors impact the most the best zone sizes found. This data is used to train several machine learning tools to predict the optimal zone sizes through the use of regression analysis. We analyze the trade-off between applicability and performance when deciding which model to use. Regardless of the choice made, our models lead to a significant improvement in order picking efficiency when compared to the arbitrary zone sizes commonly used in practice.

In Chapter 4, we study cases where correlations between product demands can be identified. This correlation means that these products have a higher chance of being picked together. Then, the storage locations can be arranged such that correlated products are stored closer to each other. This problem is modeled as a variant of the QAP and can be found in both manual and RMFS, where products are assigned to storage locations such that the interaction between pairs of products is minimized. Interactions are measured by the correlation of products and the distance between their storage locations. In this chapter, we present a powerful parallel metaheuristic to solve the QAP and four of its most famous variants. This metaheuristic extends the most successful heuristics to solve the QAP. Parallelism is used to improve solutions concurrently. The effectiveness of this method is attested by solving the hardest benchmark instances from the QAP literature. We show that our metaheuristic significantly outperforms the best methods found for all variants of the QAP.

Chapter 5 presents our study to integrate decisions commonly made when designing an RMFS. We present an integrated framework where the PRP is solved together with the OAP and the PSP in a wave picking strategy to reduce energy consumption by robots carrying the heavy pods between the storage area and the picking stations. A dynamic model is presented to plan multiple waves simultaneously when future demands are known. Since precise information about the future is seldom available, we suggest different approaches to plan waves when demands are uncertain or unknown. The integrated problem is solved using a two-stage stochastic programming model with a sample average approximation scheme to simulate future demands. Alternatively, we suggest a simple local search matheuristic to deal with larger instances. We show that integrating the PRP with the other problems leads to an increase in the picking efficiency, which is further improved solving the stochastic problem with the sampling scheme suggested.

Finally, in the conclusion, we summarize the main contributions of this thesis and points to potential future research directions.

Chapter 1

Storage and picking systems

Warehouse processes may be summarized to receiving, storage, picking, and shipping (Rouwenhorst et al., 2000). The *receiving process* checks the products when they arrive at the warehouse. The *storage process* stores them in storage locations. Products remain in the storage area until orders arrive requiring their retrieval. The retrieval of products from their storage locations is done at a *picking process*. Picked items may require sorting and consolidation activities when the batch of products is grouped according to the order requests. Finally, on the *shipping process*, the orders are checked, packed, and loaded in vehicles to be delivered.

The picking process is identified as the most labor-intensive and costly activity in a warehouse, with the cost estimated to be as much as 50–70% of the total warehouse operating expense (Frazelle, 2016; Tompkins et al., 2010). The rising of e-commerce increased the volume of operations handled in warehouses, requiring to deal with a large number of small orders, large assortment, tight delivery schedules, and varying workloads (Boysen et al., 2019a). The underperformance may result in high costs and unsatisfied customer demand (Van Gils et al., 2018c). For this reason, optimizing order picking activities are of the highest priority for productivity improvements.

The storage system consists of multiple subsystems comprising the policy used to define storage locations, the storage and the rearrangement processes at the storage area, and the technologies used for the storage system. The picking subsystems are related to the batch of orders to create the pick lists, the zoning of the warehouse, the routes followed by pickers or robots to retrieve products, and all technologies and resources involved in these operations. The interaction between the storage and picking (S/P) systems is a topic of intensive research (Dijkstra and Roodbergen, 2017; Le-Duc and De Koster, 2005; Merschformann et al., 2019; Petersen, 1999; Petersen and Aase, 2004; Tappia et al., 2019; Van Gils et al., 2018b,c).

The decisions involved in the integrated S/P system are structured at a strategic, tactical, and operational level (Rouwenhorst et al., 2000; Van Gils et al., 2018c). Strategic level decisions concern the design of the process flow and the selection of warehousing systems. A system

in which a picker follows a route inside the storage area to retrieve products contained in its pick list is known as *picker-to-product*. The most traditional one is the manual system, where the picking process is performed by human pickers (Petersen, 1997, 1999). Many assistive devices can be used in practice, such as conveyor belts, picking carts, and, more recently, augmented and virtual reality and exoskeletons (Glock et al., 2020). The opposing is the *product-to-picker* which considers that pickers stay at fixed positions and products are carried to them to be picked and separated according to the orders. Product-to-picker systems are usually associated with automated systems. Many automated systems are found, such as crane-based, shuttle-based, aisle-based, carousels, vertical lift modules, among others (Azadeh et al., 2019; Roodbergen and Vis, 2009). Recently, internet retailers are adopting robotic mobile fulfillment systems (RMFS), where robots are capable of lifting and carrying movable shelves (pods), transporting them to ergonomically designed stations where pickers retrieve the demanded products (Azadeh et al., 2019).

The choice between a manual or an automated system should take into consideration the technical capabilities of the system, such that the storage system and the suitability of the equipment used (Rouwenhorst et al., 2000). The use of automation is a means for reducing labor costs and picking times. Besides, robotic technologies provide increased flexibility and scalability due to the easiness of adding and removing pods and robots in the system (Azadeh et al., 2019). Many companies prefer to use manual systems since humans are more flexible than machines in reacting to unexpected changes in the process. For example, the variability in items' shapes and sizes is still a barrier to automation. Moreover, a large investment is usually required to automate the systems (Grosse et al., 2015; Petersen and Aase, 2004). De Koster et al. (2007) state that manual picking processes account for 80% of all picking systems in Western Europe. Independent of whether the S/P systems are mostly manual or automated, high costs are related to the storage and picking processes.

Manual warehouses have been studied for a long time, with many contributions from the managerial and optimization perspectives to several of their functions. This large body of literature led to literature reviews which focused on different aspects, such as De Koster et al. (2007); Gu et al. (2007, 2010); Reyes et al. (2019) and Van Gils et al. (2018c). Automated systems, being much more recent, have had considerably fewer studies dedicated to their specific operations. The only systematic review available that pinpoints studies dealing with the most common decision problem is da Costa Barros and do Nascimento (2021), although no discussion on the solution techniques is provided. In this chapter, we not only provide a categorization of different problems arising at the tactical and operational level of manual systems, but we also provide a classification of the problems particular to automated systems, focusing on three main categories: storage systems, batching decisions, and associated routing problems.

Regardless of the system used, the storage area in warehouses may be divided into the forward

area and the reserve area. In the forward area, products are stored in convenient units or small lots for easy retrieval by a picker. The reserve area is used to store products in the most economical way to serve as replenishment for the forward area (Bodnar and Lysgaard, 2014; Rouwenhorst et al., 2000). The decisions concerning where to locate each of these areas in the facility take into account the activity relationship between them and the other departments (De Koster et al., 2007). Here, we will focus on the decisions taken within the forward area.

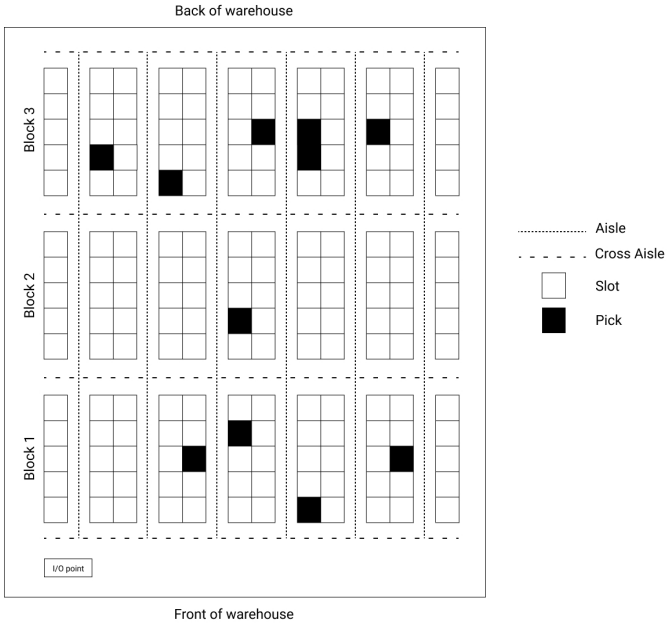
In this chapter, we consider both the manual and the RMFS and describe the most common decision problems found at the tactical and operational levels in these systems. The remainder of this chapter is organized as follows. In section 1.1, we describe the literature on manual systems, classifying them into storage, batching, or routing functions. Section 1.2 provides the same classification for automated systems, discussing the unique challenges and opportunities offered by the RMFS. Section 1.3 presents a transversal literature analysis and directions for further research. Finally, our conclusions are presented in Section 1.4.

1.1 Manual system

In manual systems, the forward area is usually organized in block layouts containing picking aisles parallel to each other, as shown in Figure 1.1. Products are stored in and picked from one or both sides of the picking aisles. The placement of additional paths within aisles, called cross aisles, separates the storage area into more blocks (Scholz et al., 2016). In a multi-block layout, a subaisle is a part of an aisle that traverses each block. The aisle height is also an important factor in the warehouse layout decisions. Wide aisles are those that allow two-way traffic of pickers and have enough space for pickers to turn around in the aisles. The space required for wide aisles is relatively high. In contrast, narrow aisles utilize less space, increasing storage capacity. However, they are less efficient since they may cause congestion as a result of blocking (Chabot et al., 2018; Van Gils et al., 2018b). Mowrey and Parikh (2014) attribute blocking to either the inability of the pickers to pass each other in the aisle due to narrow aisles (*in the aisle blocking*) or not being able to pick a product when another picker is there (*pick column blocking*). Input/output (I/O) points, also known as loading/shipping docks, pick-up/drop-off points, or depots, are placed around the storage area at points where pickers depart from and arrive to after retrieving the items in their pick lists. The operations done on the I/O points are those related to the receiving and shipping processes, such as order consolidation, packaging, and shipment. Their locations vary for each warehouse layout. Studies investigating the effect of placing them in different locations can also be found (Petersen, 1997; Petersen and Aase, 2004; Roodbergen and Vis, 2006). Other layouts, such as the fishbone and flying-V, can be potentially recommended for unit-load warehouses, but they do not offer significant benefits when performing multiple picks in a route (Gue and Meller, 2009; Ozden et al., 2020).

The decisions concerning the number of blocks, and the number, length, and width of aisles

Figure 1.1 – Layout of a multi-block storage area in a manual warehouse



in each block, should take into consideration not only the efficiency of the operations and the usage of space but also the resource constraints such as capital and available space (De Koster et al., 2007; Van Gils et al., 2018c). The criterion used is typically to maximize throughput using the minimum investment and operational costs. The throughput is characterized by productivity and the service level. The productivity can be either related to labor, picking, or equipment productivity. The service level refers to the quality of the service (e.g., the percentage of orders picked on time, or the number of pick errors). A simple way to understand the importance of throughput in a warehouse is that the sooner the order can be retrieved, considering the service level required, the sooner it is available for shipping to the customer. Consequently, the total promised shipping time can be reduced (Van Gils et al., 2018c).

Operational costs can be composed of labor and space utilization costs in manual warehouses (Mowrey and Parikh, 2014), but they are usually measured in terms of time-related performance indicators. In the picking process, this includes the setup, travel, search, and pick times (Grosse et al., 2015; Scholz et al., 2016). These four components account for 95% of the total pick time (Tompkins et al., 2010). Idle time can also be considered due to blocking or unavailability of pickers. Van Gils et al. (2018c) add the waiting, sorting, and other time-consuming activities to the other components previously mentioned. These time-consuming components can be expressed, e.g., in terms of travel cost per time unit. Travel time can also be related to travel distance. Distance has been considered as a performance criterion since the early studies on warehouse design (Christofides and Colloff, 1973). In rectangular layouts, it is usually measured using the Manhattan metric, which considers the shortest path using

the aisles and cross aisles between different points within the storage area (Theys et al., 2010).

Some constraints also considered include the target criterion (e.g., retrieve all orders within a time limit), physical constraints (e.g., the maximum height of a storage system, the capacity of the forklifts), investment constraints (e.g., maximum budget), and environmental or ergonomic conditions (Rouwenhorst et al., 2000). Constraints found in the literature include heavy products being constrained to some storage locations (Renaud and Ruiz, 2008) or to the picking sequence (Chabot et al., 2017), and the capacity of the picking device used (Gademann and Velde, 2005). The number of pickers simultaneously retrieving items in the storage area should also be considered. Single pickers may benefit from the lack of congestion while multiple pickers increase output by working in parallel. This is an important consideration while modeling order picking problems since static models are much easier to solve than dynamic ones (Klodawski et al., 2018). Finally, since humans are central actors in manual systems, ergonomic constraints play an important role. Many studies indicate that complicated routes increase memory demands, which can lead to resistance from pickers and increase the risk of missing picks (Grosse et al., 2015; Petersen and Aase, 2004).

Designing a manual system involves a large number of interrelated decisions. The most common tactical and operational level decisions are related to storage, batching, and routing. Other problems not explored in this review, such as the design of dedicated picking zones (Van Gils et al., 2018c) and the storage rearrangement (Carlo and Giraldo, 2012; Pazour and Carlo, 2015), are also important in some contexts. Detailed descriptions of these and other related problems can be found in the literature reviews of De Koster et al. (2007); Gu et al. (2007); and Van Gils et al. (2018c, 2019b). The previously described performance criteria, objectives, and constraints can be considered when solving storage, batching, and routing problems. Next, we present the decisions made at each of them and the methods used to solve them.

1.1.1 Storage

The storage location assignment problem (SLAP), as introduced by Hausman et al. (1976), consists of determining the most efficient assignment of products to locations in a warehouse to minimize the total material handling effort. Products have to be placed in convenient locations to be easily picked during the picking process. The SLAP is usually solved assuming that the set of products to be located, the set of available locations, and the distances between locations and between each location and the I/O point are known. Each slot commonly contains a single type of product, but mixed systems are also possible, such as in a gravity flow rack when multiple types of products are located in a single slot (Bodnar and Lysgaard, 2014). Usually, products are assigned to a single location, but it is possible to scatter products around the storage area if advantageous. Weidinger (2018) states that with highly heterogeneous orders, a highly scattered warehouse may lead to shorter picking routes on average because products

demanded together have a higher probability of being found near each other. Some efforts exist to estimate the picking effort when not all information is known in advance (Roodbergen and Vis, 2006; Dijkstra and Roodbergen, 2017).

The choice of a solution method to the SLAP is heavily dependent on the capacity of forecasting future demands. If future demands are too unpredictable, it is impossible to assert that a storage setup is any better than another. So, in order to design a good performance measure on the quality of a SLAP solution, it is required that some sort of forecasting exists for the arriving orders. A simple way to forecast future picks is by observing the historical order profiles. Simple techniques, such as the moving average or exponential smoothing, perform well if enough information is available at the product level (Kübler et al., 2020; van Gils et al., 2017). For products with sporadic demands, more sophisticated methods can be used, such as the nearest neighbors forecasting methodology (Nikolopoulos et al., 2016). Analytical functions, such as the ABC curve, are also typically used to account for demand skewness since it is common that few products account for the majority of the demand volume (Caron et al., 1998).

Solution methods for the SLAP are classified according to the information available about future demands. A taxonomy for the most used methods was presented by Hausman et al. (1976) and it is still widely in use. These are random storage, dedicated storage, and class-based storage.

The *random* storage policy was originally presented in Hausman et al. (1976) as the closest-open location heuristic. It considers that new products received in the warehouse are assigned to the closest empty space to the I/O point, reducing the handling effort. Among its advantages is its simplicity to use, less storage space required, and the uniform utilization of the warehouse, which reduces aisle congestion. The downside is the longer travel times for pickers to retrieve products (Chen et al., 2011; Kofler et al., 2014; Petersen and Aase, 2004). Tappia et al. (2019) affirm that when fast-moving products change rapidly over time, as in e-commerce warehouses, it is hard to implement other storage policies than the random policy. When future demands of individual products are predictable, a better alternative is to assign those with higher demands to better locations saving on average picking time.

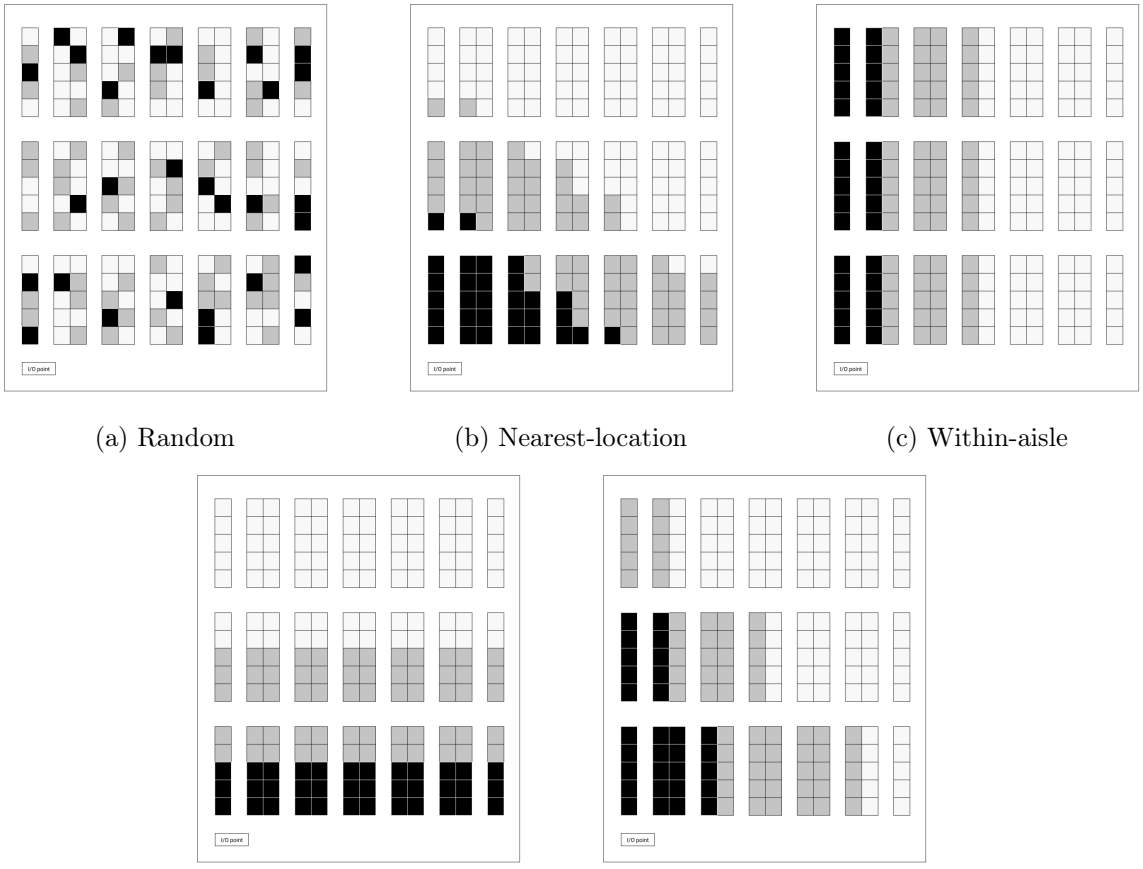
The *dedicated* storage policy, or *full turnover* policy, fixes the location of high-turnover products to most-desired locations. It consists of sorting products using a turnover-based rule and the locations using a distance-based rule and, then, assigning the most demanded products to the best locations. A major drawback of this policy is that it requires a high degree of future demand information at the product level. Also, each change in the demand rates requires a rearrangement of products in the storage area (De Koster et al., 2007). Several rules exist to sort products and locations. The cube per order index (COI) prioritizes heavy and fast-moving items to be assigned to the most desired locations (Heskett, 1963). For single picking,

COI is well known to minimize the order picking travel distance (Kofler et al., 2014). The shortest distance to the I/O point is a common rule used. Other rules based on the storage area layout can be found (Petersen, 1999; Van Gils et al., 2018b). The *nearest-location* policy, or *diagonal* policy, assigns products in a diagonal pattern from the I/O point and results in a similar rank as the shortest distance rule. The *within-aisle* policy ranks aisles according to their distance to the I/O point. In the *across-aisle* policy, products are stored across the pick aisles. For a multi-block layout, the *nearest-subaisle* policy ranks subaisles according to the distance of their heads to the I/O point. Different studies show that each of these policies performs well under different warehouse settings and all of them can be found in practice (Jarvis and McDowell, 1991; Le-Duc and De Koster, 2005; Petersen, 1999; Roodbergen, 2012).

In many cases, some sort of future demand forecasting is possible, although not with very high precision for each product. An alternative to the dedicated policy in such cases is the *class-based* storage policy. It separates products into classes, again using a turnover-based rule, and storage locations into zones then assigns each class of products to a storage zone. The final position of a product within a class is usually determined at random. Class-based storage policies are easier to implement than dedicated ones because they do not require a complete list of items ranked by volume, and they require less management time than dedicated storage policies (Petersen et al., 2004). One needs to determine the number of classes and the zone shapes and sizes. Typically, a three-zone system, known as the ABC, is preferred since it significantly improves the average route length without adding too much management complexity (Hausman et al., 1976; Larson et al., 1997; Petersen et al., 2004). For the single pick case, optimal zone shapes and sizes can be found using analytical models (Eynan and Rosenblatt, 1994; Hausman et al., 1976; Rao and Adil, 2017). However, there is no firm strategy on how to define class partitions when multiple picks are performed (De Koster et al., 2007). The zone shapes are usually defined using distance-based rules (Petersen et al., 2004). For the zone sizes, a 20/80 partition for a two-zone system, i.e., the 20% most demanded products are in the first zone and the remaining 80% in the second, and a 20/30/50 partition for a three-zone system are used as a rule-of-thumb (Dijkstra and Roodbergen, 2017; Le-Duc and De Koster, 2005; Manzini et al., 2015; Pansart et al., 2018). A few studies show that the dimensions of the zones can be defined empirically according to the skewness of the demand distribution of the products (Petersen et al., 2004; Van Gils et al., 2018b). Their findings show that the less skewed the demands are, the fewer savings a class-based policy generates in the picking process compared to the random policy. Also, higher savings are obtained in small pick lists than in larger ones. Roodbergen (2012) provides insights on the interactions between the layout, routing, and storage policies in an ABC system, showing that the best zone sizes change according to the policies in use.

Figure 1.2 provides an example of random and class-based policies for different distance rules. Squares of the same color represent products classified in the same class. Note that the

random and the dedicated storage policies are special cases of the class-based storage policy for when the number of classes is equal to one and to the number of slots in the storage area, respectively.



(a) Random (b) Nearest-location (c) Within-aisle

(d) Across-aisle (e) Nearest subaisle

Figure 1.2 – Storage policies

A different way to solve the SLAP is using a *correlated* storage policy, as introduced by Frazelle and Sharp (1989). Products that are frequently ordered together are said to be correlated. Storing correlated products close to each other may reduce the expected average travel distance. Correlation is defined as the joint probabilities that products appear in the same order. Several similarity measurements are found in the literature (Chuang et al., 2012; Jane and Laih, 2005; Xiao and Zheng, 2010). A simple and common way to calculate correlation in practice is by counting the number of times the products were picked together (Kofler et al., 2014). Then, products can be clustered and assigned to the same zones in the storage area based on their correlation (Frazelle and Sharp, 1989; Rosenwein, 1994).

A different approach to the correlated policy models it as a quadratic assignment problem (QAP), therefore not creating explicit clusters of products (Li et al., 2016; Mantel et al., 2007;

Yener and Yazgan, 2019). In the QAP, a set of products have to be assigned to a set of slots such that the interactions between pairs of products are minimized. Interactions are measured by the correlation of the products and the distance between their storage locations. The QAP requires that each product be assigned to a slot, and vice versa. In the SLAP, one extra location representing the I/O point is considered. The correlation between products and the I/O point may be defined as the product's popularity (Mantel et al., 2007; Yener and Yazgan, 2019). The QAP is considered one of the hardest combinatorial optimization problems to solve since optimal algorithms can solve only relatively small problems with a low number of products (Drezner, 2008). Therefore, heuristic methods are popular for this problem. Kim and Smith (2012) model the SLAP as a variant of the QAP and solve it using simulated annealing. Li et al. (2016) consider a dynamic SLAP, model it as another variant of the QAP, and solve using a greedy genetic algorithm. In the QAP literature, we can still find local search (Aksan et al., 2017), tabu search (James et al., 2009a; Misevičius, 2012), memetic algorithm (Benlic and Hao, 2015), ant colony (Dokeroglu and Cosar, 2016), and many hybrid metaheuristics (Dokeroglu, 2015; Dokeroglu et al., 2019; López et al., 2018b; Tosun, 2015). Also, several variants are studied, such as: the bottleneck assignment problem (Burkard, 2002), in which the largest interaction is minimized; the biquadratic assignment problem (Burkard et al., 1994), in which interactions between groups of four products are considered; the quadratic semi-assignment problem (Saito et al., 2009), in which the assumption that each slot can have only one product is relaxed; the generalized QAP (Lee and Ma, 2004), in which slots are capacitated; among others (Hahn et al., 2008b, 2010; Knowles and Corne, 2003; Punnen and Wang, 2016; Smith and Li, 2001). All the mentioned variants can be adapted to the SLAP as desired.

1.1.2 Batching

An order is the combination of several items that are requested by a customer. The retrieval of items in the storage area is done following pick lists. A pick list contains the items to be retrieved and their respective locations in the warehouse. The order batching problem (OBP) consists of determining which orders are placed on a pick list to be retrieved from their storage locations by a single picker (Petersen and Aase, 2004). Combining several orders into batches has been shown to reduce total picking time significantly. The OBP is formulated in Gademann and Velde (2005) as a set partitioning problem solved using a column generation algorithm. Due to its difficulty to solve, several heuristics are presented in the literature as batching policies. The most trivial is the *single order picking* when each pick list is composed of a single customer order (Renaud and Ruiz, 2008; Van Gils et al., 2018c). This strategy is often preferred because it is easy to implement and it reduces effort on the sorting process since order integrity is always maintained (Petersen and Aase, 2004). A similar approach is the *single item picking*, where the pick list is composed of a single item. Single item picking is mostly used in product-to-picker systems when the picker capacity or movement is limited

so that only one item can be retrieved per route. A mix of both policies may be adopted by a company. Renaud and Ruiz (2008) present a case study where these two policies are used depending on the type of customer to be served. For customers that order a large number of products, the single item picking is used, while for customers that may order any quantity of any product, the single order picking is used.

The most prominent methods to batch multiple orders are priority rule based algorithms, seed algorithms, and savings algorithms (Scholz and Wäscher, 2017). In *priority rule based* algorithms, priorities are assigned to customer orders, then orders are assigned to batches following these priorities. An example of a priority rule-based algorithm is the first-come-first-serve (FCFS) batching (Van Gils et al., 2018b). In *seed* algorithms, batches are generated sequentially starting from a selected order, called seed, determined by a selection rule. Then, accompanying orders that fit in the remaining capacity of the picker are added using another selection rule. The bin packing batching is an example of a seed algorithm (Petersen and Aase, 2004). Several others seed choice and selection rules are evaluated in Ho and Tseng (2006). *Savings* algorithms batch orders based on the time saving that can be obtained by combining such orders into one pick list. These sophisticated algorithms are based on the Clarke-and-Wright algorithm for the vehicle routing problem. A savings algorithm is used in Van Gils et al. (2018b), and it is compared to the FCFS and a seed algorithm to order batching. A description of the savings algorithm is presented in Scholz and Wäscher (2017). All these policies are designed to provide a feasible solution to the OBP. More sophisticated algorithms, such as local searches and metaheuristics can be used to improve the solutions found both for a dynamic (i.e., real-time order arrival) and static (i.e., all orders known in advance) picking (Henn et al., 2012).

If a batching is performed, a decision on the process of sorting products from different orders has to be made. The *sort-while-pick* strategy allows a picker to separate items from different orders while the picking process occurs, e.g., separating the items in different spaces of the picking equipment. This strategy is applied when orders consist of only relatively a few small items. The advantage is that no sorting process is needed after orders have been picked. Alternatively, in the *pick-and-sort* strategy, the items are collected first and sorting activities follow immediately after picking. In this strategy, the storage capacity of the picker may be used more efficiently (Gademann and Velde, 2005). However, if the additional sorting operations are too costly, any benefits gained with batching orders may quickly disappear (Petersen and Aase, 2004).

1.1.3 Routing

When receiving a pick list, the warehouse dispatches a picker from the I/O point to collect the items in the pick list and transport them back to it. The order picking problem (OPP) determines the tour to be followed by the picker to optimize a performance indicator, normally

time or distance traveled (Ratliff and Rosenthal, 1983).

Given a single pick list, the OPP can be modeled as a traveling salesperson problem (TSP) with a special cost structure related to the warehouse layout. The TSP consists of determining a minimum distance cycle that passes through each vertex once and only once. In the OPP, each location to be visited, including the I/O point, corresponds to a vertex, and the distance between two vertices is set to be the shortest distance between their corresponding locations in the warehouse. Several formulations adapted from the TSP literature can be found for the OPP (Pansart et al., 2018; Scholz et al., 2016).

Any warehouse layout can be represented using a TSP formulation. Even when special conditions apply, like for narrow aisles (Chabot et al., 2018) or scattered storage (Weidinger, 2018), the TSP formulation is adaptable to model the OPP. While the TSP is a hard problem to solve, the OPP can be solved efficiently under certain conditions. Ratliff and Rosenthal (1983) introduce an optimal algorithm for a single-block warehouse to solve it in polynomial time using dynamic programming. For a two-block warehouse, an efficient algorithm is described in Scholz and Wäscher (2017).

Since pick lists usually contain few picks, solving the OPP for a single pick list is not too expensive computationally. However, when multiple pick lists are considered, the total distance traveled is computed by solving multiple OPPs, one for each pick list. In these cases, the use of exact approaches in sequence may result in unacceptable computing times. The alternative option to solve the OPP in such cases is the use of heuristics.

Heuristics for the TSP may be used to solve the OPP as well. The most successful one is the Lin-Kernighan-Helsgaun (LKH) heuristic. The LKH approximates the optimal solutions for the OPP very well for different warehouse settings (Van Gils et al., 2018b). Other heuristics for the OPP aims to provide simple routes that can be easily memorized and accepted by pickers, minimizing the risk of a missed pick (Petersen, 1999). They are called routing policies and are chosen according to the problem characteristics, such as the warehouse shape, the number of aisles and cross aisles, the pick list size, the storage and batching policies, etc. Van Gils et al. (2018c) provide an extensive list of studies that analyzed the impact of these factors on the system performance given the routing policy chosen.

Several routing policies are found in the OPP literature. The first two policies allow pickers to enter each aisle, or subaisle for the multi-block case, only once. In the *aisle-by-aisle* policy, a picker traverses entire aisles that contain a product to be picked picking all required products before proceeding to the next aisle. The best cross aisle to move to the next aisle is determined using dynamic programming. In the *S-shape*, or traversal, policy, pickers traverse the leftmost aisle that contains picks to the back of the warehouse and then return to the front picking products one block at a time. Any subaisle containing a pick is traversed. After the last pick in a block, the picker returns to the front of that block and continues to the next. In a

single-block layout, aisles containing picks are entirely crossed. A return is allowed in the last pick of the last aisle visited if it is advantageous.

The next three policies delimit a point in each aisle or subaisle where a picker cannot pass through. The steps are taken for a single-block warehouse layout, but they can be adapted to a multi-block case. The *return* policy considers that each picker enters and leaves through the same end of aisles containing pick items, i.e., the limit point is the deepest pick in the aisle. In the *midpoint* policy pickers can go as far as the middle of the aisle before returning to leave it from the same end they entered. Exceptions are applied only for the first and last aisles containing picks so that the picker can access the opposite cross aisle and return to the I/O point after all picks. In the *largest gap* policy, pickers enter aisles as far as the start of the largest distance between two adjacent picks in that aisle. They may also opt to avoid traversing it if the distance between the first or last pick and the closest end of the aisle is larger. Then, they return to leave at the same end that they entered. As in the midpoint policy, a picker never crosses the entire aisle except in the first and last aisles containing items. Rao et al. (2020) present a closed-form expression for the expectation of the largest gap between consecutive picks in an aisle.

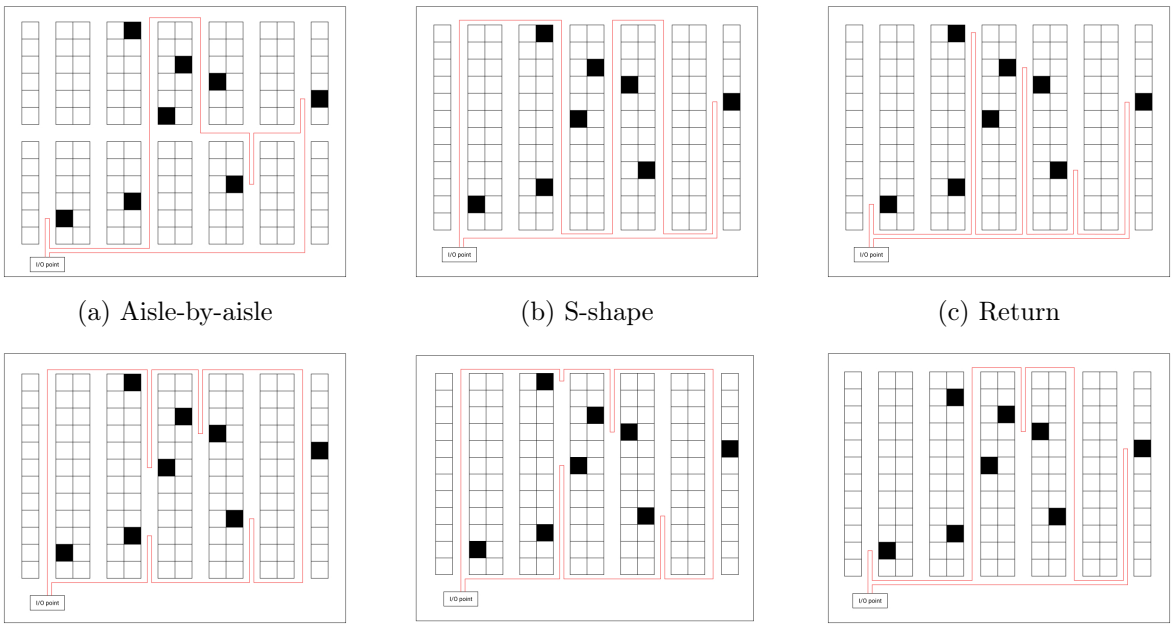
The *combined* policy combines the two types of decisions described before, i.e., entirely crossing aisles or returning to leave at the same end. The picker enters an aisle as far as the deepest pick in it. Then, a decision on either going to the end of the aisle or returning to exit at the same end entered has to be made. The choice is made using dynamic programming by always looking one aisle ahead to be at a better starting point for the next aisle. An example of all six routing policies presented is shown in Figure 1.3. A detailed description of all steps considered in these heuristics can be found in Petersen (1997) for the single-block layout and in Roodbergen and De Koster (2001) for the multi-block layout.

The pick list size is a factor that greatly influences the performance of routing policies. Petersen and Aase (2004) observe that for larger pick lists, the optimal routing tends to form S-shape routes, no matter the SLAP and the OBP policies used. Also, as the average order size increases, the use of a more complex routing policy yields minimal improvements. This is an interesting observation because larger pick lists are harder to solve to optimality in non-standard warehouses.

1.1.4 Interactions between the storage, batching, and routing problems

Since the SLAP, OBP, and OPP are mainly solved using heuristic policies, many studies try to find which combinations of policies perform best under different warehouse settings. Van Gils et al. (2018c) survey studies that investigate the combination of policies for these and other problems found in the manual system design.

The interactions between the SLAP and the OPP are also well studied. One of the earliest



(a) Aisle-by-aisle (b) S-shape (c) Return

(d) Midpoint (e) Largest gap (f) Combined

Figure 1.3 – Routing policies

studies is presented in Petersen (1999). He analyzes the performance of combined, largest gap, and S-shape routing policies with within-aisle and diagonal dedicated storage policies. He concludes that the combined and largest gap policies are good performing combinations with the diagonal policy. S-shape combines better with within-aisle, although the other two routing policies performed better than S-shape even for within-aisle. Van Gils et al. (2018b) investigate the effect of other pairs of storage and routing policies concluding that the optimal routing policy significantly outperforms all routing heuristics in all combinations of layouts tested. Within-aisle also combines well with the aisle-by-aisle and the optimal routing (Petersen and Aase, 2004; Theys et al., 2010). Another good combination observed is the return with across aisle (Caron et al., 1998). For a class-based storage policy, Petersen et al. (2004) show that savings are attainable using either optimal or S-shape routing compared to random storage, but they are higher when a within-aisle policy is used. A different approach is to model the SLAP considering imposed routing policies. Mantel et al. (2007) assume a specific warehouse system (vertical lift module) and solve the SLAP for an S-shape policy. Also, Dijkstra and Roodbergen (2017) use dynamic programming to optimally assign products for several heuristic routing policies, showing the efficiency losses observed from the deviation of solutions obtained using their method and the common heuristic storage policies.

The SLAP and OBP interactions are also investigated, although less intensively. Most studies only consider random and within-aisle storage in combination with the batching policies (Van Gils et al., 2018c). Seed rules that minimize the number of aisles are preferred with a within-aisle dedicated storage policy (Ho and Tseng, 2006). Other more sophisticated meth-

ods, such as metaheuristics, also perform well with within-aisle (Henn and Wäscher, 2012; Henn and Schmid, 2013; Hsieh and Huang, 2011).

In their review, Van Gils et al. (2018c) show that the OBP and OPP is the combination of S/P problems with the highest number of papers found that investigate their interactions. These problems can be solved in an integrated manner using mathematical programming models, mainly for static picking. Due to the complexity of solving these models exactly, it is common to solve them using (meta-)heuristics (Aerts et al., 2021; Briant et al., 2020; Cheng et al., 2015; Li et al., 2017; Lin et al., 2016; Van Gils et al., 2019a). For the heuristic policies, Van Gils et al. (2018b) observed that when the FCFS and the savings batching policies are used, the largest gap and S-shape routing policies perform best. Meanwhile, when the seed batching policy is used, the return routing policy performance increases significantly.

Finally, the interactions between the joint SLAP, OBP, and OPP are investigated in Kübler et al. (2020) where metaheuristics are proposed to solve these three problems in a dynamic context.

1.2 Robotic mobile fulfillment system

The RMFS is a relatively new category of automated S/P system (Guizzo, 2008). It was popularized by Kiva Systems Inc., later acquired by Amazon, which largest warehouses control thousands of robots (Fragapane et al., 2021). Since then, other providers have entered the mobile robots market (Lamballais et al., 2020; Weidinger et al., 2018). The robots in an RMFS are battery-powered automated guided vehicles that are guided by bar codes on the floor to orientate them (Guizzo, 2008). They can lift pods with more than 1,000 kg and move at a human walking speed (da Costa Barros and do Nascimento, 2021). A new generation of autonomous robots, which do not require fixed paths in predefined points to move, is being introduced, adding flexibility to the operating environment (Fragapane et al., 2021).

In addition to the intrinsic advantages associated with automatization, an RMFS provides increased flexibility and scalability due to the easiness to add and remove pods and robots in the system. They also require a relatively low investment cost even for a large fleet of robots compared to other automated systems. Throughput rates are much higher in an RMFS compared to manual warehouses. Due to the lifting capacity of robots, RMFS are efficient when used in warehouses containing several small and lightweight items, which is a perfect fit for e-commerce warehouses (Wulfraat, 2012; Boysen et al., 2019a).

The workflow in an RMFS is as follows. Products arriving at the warehouse or coming from the reserve area are replenished into identical storage shelves, called inventory pods, with empty storage slots brought by robots to the replenishment stations. Then, the robots move the pods to an empty storage location into the storage area where they wait to be requested

again. Once requested, a robot moves underneath the pod, lifts it, and brings it to a picking station, where a stationary picker retrieves the products demanded (Boysen et al., 2017).

The storage area has a grid format, and each square represents either an aisle or a storage location, either containing a parked pod or not. Replenishment and picking stations are located around the storage area. Each space in the grid can be occupied by a single robot at a time, and robots may travel under pods if required. A buffer zone exists between the stations and the storage area where robots with pods can form a queue. An example of a RMFS floor plan is presented in Figure 1.4. The aisles in an RMFS are typically one-way to avoid congestion. Two-way aisles are mostly used when robots are aisle-captive, i.e., dedicated to a designated aisle (Wang et al., 2020). Recharging stations for the robots are usually located around the storage area. Other layouts can also be found, although they are less common in practice (Aldarondo and Bozer, 2020; Jin et al., 2020).

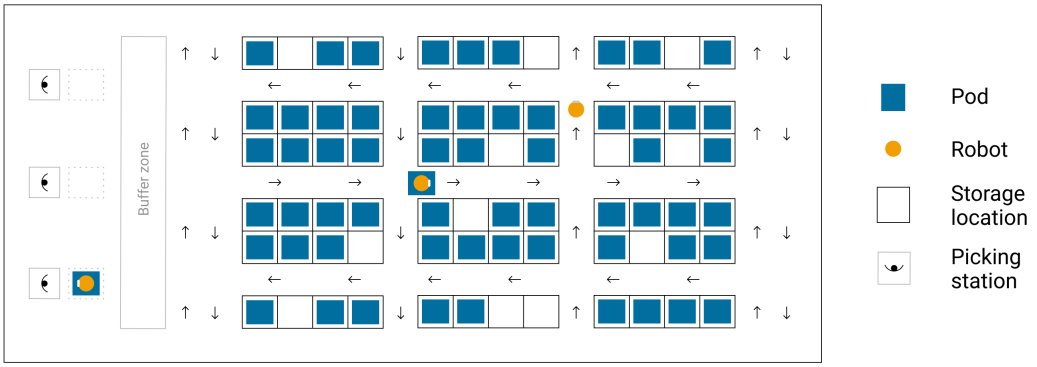


Figure 1.4 – Representation of an RMFS storage area layout

Many performance measures are considered when evaluating the picking process in an RMFS. These are usually analyzed in the literature through the use of queuing models to simulate the dynamic arrival of orders and the movement of robots in the storage area. Azadeh et al. (2019) classify these studies as "system analysis" since their focus is on modeling techniques to estimate the performance of the system. The most common measure is the throughput rate, usually given as the rate of orders treated per hour. Lamballais et al. (2017) estimate the maximum order throughput, average order cycle time, and robot utilization of an RMFS with different settings. Zou et al. (2017) analyze different policies for assigning robots to stations to evaluate the retrieval throughput time. Yuan and Gong (2017) estimate the optimal number and velocity of robots to maximize throughput time. Zou et al. (2018) estimate system performance under different battery charging and swapping strategies, finding that inductive charging allows a higher throughput time. Yuan et al. (2018) estimate the throughput rate for different replenishment policies. Roy et al. (2019) analyze the effects on system throughput for different assignment strategies for robots to storage zones considering both the picking and replenishment processes. Lamballais et al. (2020) evaluate the optimal number of pods

per product, the ratio of picking/replenishment stations, and the replenishment level per pod that improves throughput. Wang et al. (2020) suggest using aisle-captive robots to improve throughput and average order flow time. Gong et al. (2020) calculate the throughput of an RMFS where orders are classified by their urgency determining the optimal number of robots and stations. A simulation framework for the RMFS is provided by Merschformann et al. (2018). It is used later in Merschformann et al. (2019) to analyze the trade-offs between seven performance measures for different policies combinations used to solve the most common operational problems found in an RMFS.

Following the structure suggested by Azadeh et al. (2019), another type of study found is on "operations planning and control". Studies in this category focus on optimizing the system rather than simply evaluating it. Most of these studies consider problems within the RMFS and optimize them, generally using mathematical programming and heuristics. Although throughput measures can also be optimized, in this type of study the focus is on optimizing different performance indicators. Examples are minimizing the distance traveled by robots (Gharehgozli and Zaerpour, 2020; Weidinger et al., 2018), the total energy consumed (Li et al., 2020), or the total number of pod visits to picking stations (Boysen et al., 2017; Jiang et al., 2020; Valle and Beasley, 2021; Xiang et al., 2018; Xie et al., 2021). In a different approach, Aldarondo and Bozer (2020) develop analytical formulas to estimate the expected travel distances and throughput capacity as a function of the order assignment rule, the shape of the forward area, and the location of the picking stations.

Even before its popularization, Enright and Wurman (2011) discussed several decision problems which could be considered for future research to optimize an RMFS. A literature review on recent developments on the RMFS is provided in da Costa Barros and do Nascimento (2021). As for the manual system, we are interested here in describing and analyzing tactical and operational decisions for storage, batching, and routing. Merschformann et al. (2019) provide a classification for the most common problems found to design an RMFS. Next, we present the decisions made at some of these problems and the methods used to solve them.

1.2.1 Storage

Storage location decisions in an RMFS are divided into two distinct categories. The first one consists of choosing the assignment of products to pods during replenishment activities. Different than in manual systems, scattered storage is widely used in RMFS. This storage policy reduces waiting time in an environment with multiple picking stations. This is because if a popular product is stored in a single pod that pod may be very busy moving between different stations, which could delay the picking process (Boysen et al., 2017; Valle and Beasley, 2021). The scattered storage also increases the probability of having pods containing multiple products to be picked together. The "pile-on" measures the average number of picks per pod visit, and it is also a common performance measure optimized in an RMFS. To improve the

pile-on, several studies use correlated-based models to assign correlated products to the same pod. As for the manual system, these models can be based on a QAP variant (Xiang et al., 2018), clustering, (Li et al., 2020; Mirzaei et al., 2021) or other heuristic policies (Guan and Li, 2018; Yuan et al., 2018).

The second category of storage location decisions is represented by the pod repositioning problem (PRP) (Xie et al., 2021). The PRP consists of determining where to park a pod in the storage area after being treated at a replenishment or picking station. When a pod is returned, it can be repositioned in any empty storage location. Several heuristic policies can be used to solve the PRP, most are equivalent to the policies used for the SLAP in manual systems.

In the *random* policy, the pod is returned to any free storage location. The *nearest* policy determines that a pod should return to the nearest empty location to the station where the replenishment or picking was last performed, seeking an improvement in the space utilization. In the *fixed* policy, the pod always returns to the same location it was previously assigned. Usually, a turnover-based rule is applied to decide the fixed locations of the pods (Aldarondo and Bozer, 2020; Li et al., 2020). Although this policy is commonly used in manual systems, in an RMFS it is more advantageous to reposition the pod in a more convenient location dynamically to save time and energy. A related method is the *class-based* policy, which assigns pods to fixed zones within the storage area (Wang et al., 2020). The exact position within the zone where the pod is parked can be defined using, for example, the random or nearest policies. Weidinger et al. (2018) suggest considering when the pod will be required again, if this information is available, to determine whether it is positioned closer or further away from the station to be visited next. Following this idea, they suggest the *shortest path* policy, which assigns the pod to a storage location such that it results in the shortest route considering the previous station visited and the next one. When the set of future pod requests is known, an *optimal* policy can be modeled as a scheduling problem as in Weidinger et al. (2018), or as a TSP as in Gharehgozli and Zaerpour (2020). In both cases, the total travel distance followed by a robot to perform all tasks is minimized. Rimélé et al. (2021) still suggest solving the PRP dynamically since the state of storage locations, either being occupied or empty, is uncertain due to the stochastic travel times of robots.

1.2.2 Batching

The batching problem in an RMFS defines how to assign orders to stations, called the order assignment problem (OAP) (Merschformann et al., 2019). In the OAP orders are usually assigned to stations in real time to improve the system throughput. So, a decision on the order sequencing is relevant (Boysen et al., 2017; Valle and Beasley, 2021). The OBP for a manual system differs from the OAP since orders are batched before being assigned to a picker, improving the system efficiency.

In an RMFS, once an order arrives, it is placed in a backlog containing all orders not yet assigned to stations. The capacity of a station is defined as the number of orders or products it can handle at a time. In a busy environment, where several orders are available in the backlog, all the station capacity is used and, whenever an order is finished, another one from the backlog has to be assigned to that station. Merschformann et al. (2019) present several policies to solve the OAP. In the *random* policy, a random order from the backlog is assigned. The *FCFS* policy assigns the order received first to reduce throughput times. The *due-time* policy selects the order with the earliest due time, aiming to finish orders before their deadline. The *common-lines* policy compares the currently assigned orders with all orders from the backlog and selects the one with the most products in common with the objective of increasing the pile-on. The *pod-match* policy selects the order from the backlog that best matches the pods heading to the station at the moment of assignment. Merschformann et al. (2019) show that the pod-match policy performs best since it looks at the incoming pods at a station. A modified version of this policy is used in Xie et al. (2021) where a pick-and-sort strategy is adopted by considering that orders can be split between several stations. This can increase the picking efficiency at the cost of requiring additional consolidation time during the packing process.

A growing number of papers suggest wave picking in an RMFS as a way to minimize the number of pod visits. In wave picking, the trigger to solve the batching problem is the end of the current wave. Thus, multiple orders can be batched and assigned together to stations. Larger batch sizes are preferred to an energy-efficient picking (Xu et al., 2019). Usually, a balanced workload among pickers is considered, such that approximately the same number of picks is performed in each station in each wave (Valle and Beasley, 2021). The batching problem is formulated such that a known set of orders is separated into different batches and each one is assigned to a station. Due to the balanced workload, it is expected that all stations finish their job almost simultaneously. Then, the next wave of batched orders is assigned to the stations, and so on. Variants of this problem are formulated and solved in Boysen et al. (2017); Jiang et al. (2020); Valle and Beasley (2021); Xiang et al. (2018).

1.2.3 Routing

Since pickers are stationary, the routing problem in an RMFS considers the paths traveled by robots within the storage area. Robots cycle between picking pods, carrying them to stations, returning them to empty storage locations, and moving to the next pod demanded. This is called a dual command cycle. Each of these trips follows a path computed using any shortest-path algorithm, such as the A^*, for a collision-free path considering other robots' locations and static obstacles on the floor (Kumar and Kumar, 2018; Li and Liu, 2016). Although routing costs can be measured in function of the total distance traveled by robots, a better measure is to convert it to energy consumption, since in an automated warehouse the electricity costs

are prevalent (Pazour and Carlo, 2015). Models that estimate the energy consumed by robots in operation in an RMFS are found in Li et al. (2020) and Zou et al. (2018).

Once the storage and batching problems are solved, a task is sent from a central server to robots to retrieve the pods that contain the products demanded. The particular robots chosen have to be defined. The task allocation is optimized by considering the arrival sequence of tasks and the position of robots (Li and Liu, 2016; Zhou et al., 2014).

Due to the scattered storage, deciding which pod to select for transportation is also a decision in an RMFS, known as the pod selection problem (PSP). Merschformann et al. (2019) also present several policies to solve the PSP. The *random* policy selects a pod that contains at least one product in the orders assigned to the station. The *nearest* policy selects the pod located in the nearest position to the station according to the path planning solution. The *pile-on* policy selects the pod containing the most products demanded in the orders assigned to the station. The *demand* policy selects the pod containing the most products demanded in the backlog. The *lateness* policy aims to fulfill late orders by selecting pods with products needed to fulfill them. Finally, the *age* policy aims to finish the oldest orders of a station by selecting a pod containing the products needed to fulfill the oldest order.

1.2.4 Interactions between the storage, batching, and routing problems

The interactions between the PRP, OAP and PSP are investigated in Merschformann et al. (2019). After a series of experiments, they observe some cross-dependencies between different combinations of policies used to solve these problems. For example, the demand policy for the PSP is among the best and worst performing combinations of policies depending on the other policies used. Aldarondo and Bozer (2020) study the expected dual command distances as a function of the PRP and PSP policies combinations, the shape of the storage area, and the configuration of the stations around it. Two combinations of policies are investigated: random PRP with random PSP and fixed PRP with nearest PSP. They show that the latter yields an expected route length up to 65% lower than the former.

Several studies propose the integration of the OAP and PSP. After deciding the allocation of products to pods, Xiang et al. (2018) model the OAP as an OBP in which a predefined number of batches have to be created to minimize the number of pods visits. Jiang et al. (2020) propose a model for wave picking where replenishment decisions are integrated with the OAP and PSP. Xie et al. (2021) propose another model where decisions are made periodically considering the pods currently available in the stations and that orders can be split among stations or split over periods. Valle and Beasley (2021) propose a model that integrates the OAP and PSP and, then, the sequence that the pods visit the stations is defined. Finally, Zhuang et al. (2021) integrate the OAP and PSP with the order and pod sequencing problems using a linear programming model and solve large instances using a metaheuristic. No study was found that

integrated these two problems with the PRP.

1.3 Literature analysis and research directions

The previous sections introduced some of the most commonly solved tactical and operational problems for the planning of manual and RMFS warehouses. For the manual system, the most observed effort in the literature is undoubtedly the analysis of the interactions between the problems. On their review, Van Gils et al. (2018c) found 61 papers dealing with combinations of these planning problems, mostly published in the last decade, which shows the strong recent trending of such publications. For the RMFS, however, the literature is still too recent and little attention has been given to analyze the interactions between multiple problems.

The integration of problems is even less studied. For both systems, integrated decisions are mostly for the joint routing and batching problems. Although manual systems are being studied for decades, almost all integrated approaches found by Van Gils et al. (2018c) were published in the last years. This is mainly because integrated models are usually hard to be solved since they may result in nonlinear or stochastic programming models.

Heuristics as an alternative for optimal approaches are found. The most commonly used are the simple policies presented here, which are proposed to deal with each problem individually. Since most studied problems are assignment problems, the pairwise exchange local search is also commonly used to improve solutions. It is found at least in Ho and Tseng (2006); Renaud and Ruiz (2008); Scholz and Wäscher (2017); and Xiang et al. (2018). We also found many metaheuristic approaches used to diversify the search for the optimal solution. They include tabu search (Chen et al., 2011), simulated annealing (Ho and Tseng, 2006; Kofler et al., 2011, 2014; Pazour and Carlo, 2015), genetic algorithm (Carlo and Giraldo, 2012; Guan and Li, 2018), variable neighborhood search (Jiang et al., 2020), and adaptive large neighborhood search (Chabot et al., 2018; Gharehgozli and Zaerpour, 2020; Weidinger et al., 2018; Zhuang et al., 2021).

From this literature analysis, we summarize the research opportunities observed as follows:

- **Integration within storage decisions**: most of the studies integrating warehouse design planning problems consider only batching and routing decisions. The integration of these with storage decisions is important due to the interdependencies of these problems;
- **Different performance measures**: Most of the formulations found for manual systems are focused on optimizing the same performance measures (time or distance). Other measures, such as pick accuracy and service level, require more investigation. For example, scattered locations could increase the probability of pick errors due to the increasing complexity of the storage area setup. For the RMFS, most studies focus on maximizing

order throughput rates without considering the implications on operational costs. More attention should be given to better evaluate the efficiency of this system, for example, measured by the energy consumption of robots;

- **S/P systems comparison**: the comparisons between the manual system and the RMFS found are mostly qualitative. No study directly compares the trade-offs between an increased order throughput provided by the RMFS with the costs to implement it. Future studies should provide clear guidelines on when it is more advantageous to change from one to another, which can help managers to adopt the use of automated vehicles in other distribution centers than for e-commerce;

- **Congestion effects**: Warehouses are very dynamic environments, and studies that consider congestion in the manual system literature are still scarce. The analysis of storage and picking processes without considering congestion effects can lead to misleading conclusions in most modern warehouse systems. So, the effect of congestion on operational decisions and performance measures is another topic with great opportunities to research;

- **Human factors and new technologies**: despite the increasing automation of S/P systems, they still rely largely on manual human work. The interaction between humans and new technologies in S/P systems is a topic largely neglected. The impacts of the introduction of new technologies in warehouses, such as virtual reality, augmented reality, exoskeletons, and automated vehicles, should be more investigated. It is still unclear how they will affect the performance of the pickers and of the S/P systems overall;

- **New optimization methodologies**: The advances in computational power and new business analytics methodologies open an opportunity to reinvestigate classical optimization problems found in warehouses using more powerful tools. Parallel computing, machine learning, and stochastic optimization are some of the fields offering novel tools that can be used to solve problems otherwise considered infeasible to be modeled and solved;

- **Adapt classical methods to modern systems**: Although the RMFS is a relatively new system, many techniques used to solve problems for other systems can be adapted to it. Studies are showing that storage problems in an RMFS can be solved efficiently using storage policies for the manual system. The same can be said about routing and batching. Future research should account for the advantages and disadvantages of using classical solution methods against most sophisticated ones, thus providing convincing arguments to managers and pickers to accept the changes.

1.4 Conclusions

This chapter addressed the most common tactical and operational problems in the warehousing literature to design manual and robotic mobile fulfillment systems. Specifically, decisions related to storage, batching, and routing are described and a brief overview of studies that investigate them is provided.

The problems investigated have been shown to have interdependencies not yet fully investigated. They are usually solved sequentially and their interactions are evaluated for different solution approaches, such as for simple policies. The recent development of sophisticated optimization tools and the improvement in computational capacity is opening new research opportunities to investigate problems that otherwise could not be solved. For this reason, the number of studies proposing to integrate them using a single and general framework is growing. In this thesis, we use many modern techniques to integrate and solve storage, batching, and routing problems found in the manual and RMFS.

Chapter 2

Integrating storage location and order picking problems in warehouse planning

Chapter information A paper based on this chapter is published in *Transportation Research Part E: Logistics and Transportation Review*: A. Silva, L. C. Coelho, M. Darvish, and J. Renaud. Integrating storage location and order picking problems in warehouse planning. Transportation Research Part E: Logistics and Transportation Review, 140:102003, 2020.

Résumé

Dans ce chapitre, les problèmes d'affectation des produits et de prélèvement des marchandises dans un système manuel d'entreposage sont intégrés. Dans ce problème, nous voulons affecter des produits à des emplacements de stockage afin de minimiser la distance totale parcourue pour cueillir l'ensemble de produits commandés. Nous avons modélisé le problème intégré et d'autres quatre cas particuliers qui imposent des politiques de routage (*retour*, *forme de « S »*, *point milieu* et *l'intervalle le plus grand*) pour l'arrangement en un seul bloc. Les modèles sont linéarisés à l'aide de techniques de linéarisation simples. Nous résolvons ces modèles à l'aide d'un solveur commercial pour des instances artificielles qui simulent les entrepôts couramment rencontrés en pratique. Les résultats montrent que ces modèles sont limités à résoudre des problèmes à petits entrepôts et à peu de commandes. Comme alternative, nous présentons une métaheuristique générale de recherche de voisinage variable qui s'avère très efficace pour les petites instances. Pour des entrepôts plus grands et avec plus de prélèvements, nous montrons que notre métaheuristique améliore considérablement les solutions générées par les politiques de stockage communes.

Summary

In this chapter, we investigate the integration of the storage location assignment and the order picking problems for a manual warehousing system. The objective of the problem is to assign products to storage locations to minimize the total route length traveled by pickers to fulfill a set of orders known. We modeled the integrated problem and four special cases for the single-block layout with imposed routing policies (*return*, *S-shape*, *midpoint*, and *largest gap*). The models are linearizes using simple linearization techniques. We perform computational experiments with a commercial solver on instances generated to simulate warehouse settings commonly found in practice. The results show that the effectiveness of these models is limited to small warehouses and few orders. As an alternative, we present a General Variable Neighborhood Search metaheuristic, which is observed to be very efficient for those small instances. For larger warehouses and with more picks, we show that our metaheuristic significantly improves solutions generated by common storage policies.

2.1 Introduction

The most resource-intensive process performed in a warehouse is the order picking activity (Scholz and Wäscher, 2017), which is highly dependent on the storage location policy used. The storage location assignment may be performed immediately after the reception of a product or periodically. Its purpose is to place products in convenient locations where they can be easily picked during the picking process. Order picking refers to the activities performed to retrieve products from their storage locations to satisfy demands specified by customer orders. Due to labor intensity in such systems, the order picking process alone concentrates 50 to 75% of the total operating costs for a typical warehouse (Frazelle, 2016). Thus, warehouses are often optimized for cost-efficient order picking.

The problem of deciding where to locate products in a warehouse is known as storage location assignment problem (SLAP). It consists of determining the most efficient assignment of products to locations in order to minimize the total handling effort. The order in which the products are collected is determined by a routing strategy to be followed by the pickers in a problem known as order picking problem (OPP). The SLAP solution is an input of the OPP, since routes can be created only after product locations are known. At the same time, a SLAP solution can only be evaluated when the strategy to solve the OPP is known.

The SLAP and OPP as individual problems have been extensively studied (De Koster et al., 2007; Gu et al., 2007). When considered together, they are usually solved sequentially for different combinations of storage location and picking policies. These policies are usually very simple heuristics used to create a solution for each problem. They present advantages such as being easily memorized, reducing costs caused by pick errors, and they are easily accepted by the pickers (Petersen and Aase, 2004; Petersen et al., 2004).

Traditionally, SLAP is considered to be a tactical problem, while the OPP is an operational one. However, since the warehouse environment is dynamic, there is a need to continuously shift the location assignments of products. In a dynamic slotting strategy, the storage area of a warehouse is reorganized more frequently. Reassignments can occur, for example, on a daily (Kim and Smith, 2012) or monthly (Kofler et al., 2014) basis specifically for each pick wave or "period". When multiple periods are considered, the storage area can be reorganized between each period in order to minimize the total travel distance by all pickings and reassignments. SLAP solutions are evaluated by calculating the exact routes that will be performed to retrieve the products demanded during a period. Thus, the storage allocations may be updated more frequently in order to optimize pickings performed during the considered period. Nevertheless, little attention had yet been given to integrating these two problems in warehouse planning (Van Gils et al., 2018c).

In this paper, we integrate the SLAP and OPP and solve them as a single problem that we name as *storage location and order picking problem* (SLOPP). We present a cubic mixed integer programming (MIP) model to solve the general integrated SLOPP, which does not assume any specific warehouse layout. However, we cannot ignore that heuristic routing policies are still widely used, so we also present MIP models to solve non-integrated versions of the SLOPP. In these versions, the picking is performed using well known heuristic routing policies as return, S-shape, midpoint and largest gap. We name these non-integrated versions, respectively, SL+Re, SL+Ss, SL+Mp and SL+Lg. As special cases of the integrated SLOPP, a feasible solution for any of them is clearly a feasible solution for the SLOPP itself. The four special cases are mathematically modeled considering the most common warehouse layout found in practice, i.e., the rectangular single-block layout with multiple aisles. We also propose a General Variable Neighborhood Search (GVNS) metaheuristic framework to approximate optimal solutions for the SLOPP and its special cases. Computational experiments attest to the effectiveness of the GVNS in solving the problems for the small instances. We also test it for regular size instances, as used in literature, to analyze the advantage of solving the integrated version of the problem. We show that optimizing storage location assignments can yield large gains in the picking performance when compared to arranging the storage locations using common storage policies, regardless of which routing policy is used to solve the OPP subproblem.

The remainder of this paper is organized as follows. In Section 2.2, we review the literature of the SLAP and the OPP. The MIP models for the general and integrated SLOPP and its four special cases are presented in Section 2.3. The proposed GVNS is presented in Section 2.4. Section 2.5 provides the computational experiments results. In Section 2.6 we discuss the practical implications of this research. Finally, our conclusions follow in Section 2.7.

2.2 Background on the storage location assignment and order picking problems

We present a brief review on the SLAP and OPP. For complementary information, we refer to the literature reviews in De Koster et al. (2007); Gu et al. (2007, 2010); Reyes et al. (2019) and Van Gils et al. (2018c).

2.2.1 Storage location assignment problem

Problems that involve the assignment of products to storage locations in the warehouse are known as SLAP. Several different performance measures to be optimized and constraints can be considered. The most commonly used is associated with the space and distance, which includes the minimization of travel distances (Reyes et al., 2019). This problem considers the storage area layout, the set of orders to be fulfilled and assumes a picking policy. The SLAP was introduced by Hausman et al. (1976) for an automated warehouse, and it has been widely studied since then (e.g., Carlo and Giraldo (2012); Kofler et al. (2014)).

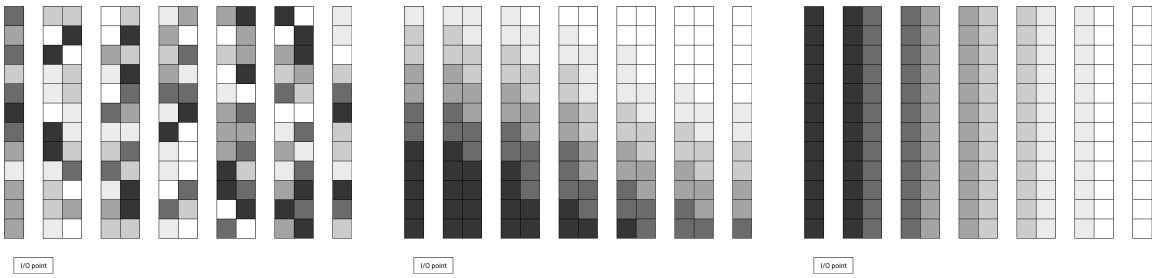
The SLAP is related to the quadratic assignment problem (QAP), which is considered to be one of the hardest combinatorial optimization problems to solve optimally due to its nonlinear objective function (Drezner, 2008). In their review, Reyes et al. (2019) report that although exact methods exist to solve SLAP variants, such as those based in clustering and affinity-based assignment techniques (e.g., Chuang et al. (2012); Li et al. (2016); Ming-Huang Chiang et al. (2014)), they usually are not used due to the complexity of the problem.

Since SLAPs are very hard problems to solve using exact methods, typically simple heuristics are used to generate the product assignments. These heuristics are typically divided into the random, dedicated, and class-based categories. This taxonomy was presented by Hausman et al. (1976) and it is still widely in use. The choice of the most appropriate policy depends on the available product information (Carlo and Giraldo, 2012; Gu et al., 2007).

The *random storage* consists of the random assignment of products to locations in the storage area. Its main advantage is its simplicity, but the downside is obviously the longer travel times for pickers to retrieve products (Chen et al., 2011; Kofler et al., 2014; Petersen and Aase, 2004). Tappia et al. (2019) also point that when fast moving products change rapidly over time, like in e-commerce warehouses, it is hard to implement other storage policies than the random one. The *dedicated storage* policy fixes the location of high-turnover products to "best" locations. Products are sorted using a demand based rule and locations sorted based on the distance to the I/O point and, then, the best products are assigned to the best locations. Several rules used to sort products and locations are discussed in literature (Kofler et al., 2014; Petersen, 1999; Van Gils et al., 2018b). The *class-based storage* policy is an alternative for when demand forecasting is possible but not very precise for each individual item. The class-

based policy divides products into classes and storage locations into zones, then assigns each class to a zone. The final position of a product within its zone is usually determined randomly. A different way to classify items is by their affinity. Items that are frequently ordered together are said to be affine. Some studies (Mantel et al., 2007; Yener and Yazgan, 2019) use the QAP formulation to allocate products based on affinity, although not creating explicit classes. References for studies that use affinity-based rules in the classification of products are found in Kofler et al. (2014).

Figure 2.1 provides an example of random and class-based policies (diagonal and within-aisle metrics). Squares with the same background color represent products with similar demand. We note that the random and the dedicated storage policies are special cases of the class-based one for when the number of classes is equal to one and to the number of products to be located, respectively.



(a) Random (b) Diagonal (c) Within-aisle

Figure 2.1 – Storage policies

2.2.2 Order picking problem

The objective of the OPP is to optimize a performance measure, e.g., minimize time or distance traveled by the picker. For pick lists with only one product, the OPP optimal solution is easily generated by solving a shortest path problem for the round trip between the I/O point and the product location. For a pick list with products in multiple locations, an optimal OPP solution is obtained by solving a special case of the Traveling Salesman Problem (TSP) (Applegate et al., 2007), given that there is enough capacity for the picker to retrieve all products in only one route. In the OPP, each item in the pick list, as well as the I/O point, corresponds to a vertex and the distance between two vertices is set to be the shortest distance between their corresponding locations in the warehouse. Formulations for the OPP adapted from the TSP literature are found in Pansart et al. (2018) and Scholz et al. (2016).

Any warehouse layout can be represented using a TSP formulation. Even when special conditions apply, like for narrow aisles (Chabot et al., 2018) or scattered warehouses (Weidinger, 2018). While the TSP is a hard problem to solve, the OPP can be solved efficiently under cer-

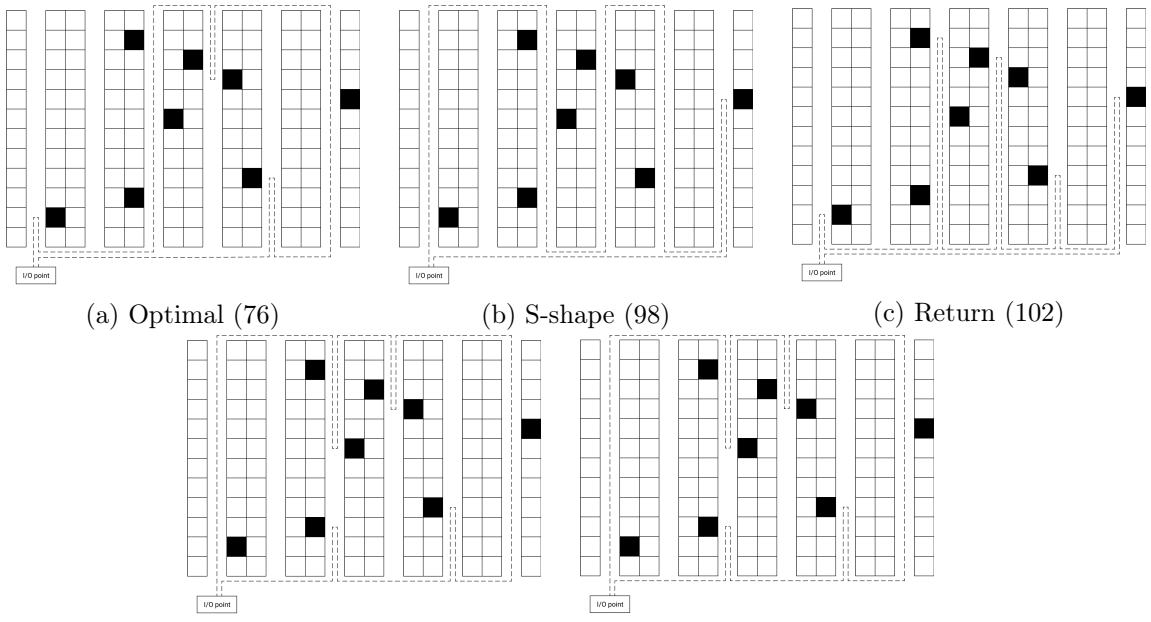
tain conditions. Ratliff and Rosenthal (1983) introduce an optimal algorithm for a rectangular single-block warehouse which is solvable in polynomial time using dynamic programming. The two-block case is described in Scholz and Wäscher (2017). Heuristics used for the TSP may be used to solve the OPP as well. The most successful one is the Lin-Kernighan-Helsgaun (LKH) (Helsgaun, 2000). Theys et al. (2010) used LKH to approximate optimal OPP routes reporting an average optimality gap of 0.1% for different warehouse settings.

While LKH searches for near-optimal OPP solutions, certain routing policies provide simple solutions that can be easily memorized and executed by the pickers. These routing policies are chosen according to the problem characteristics, such as warehouse shape, number of aisles and cross aisles, pick list size, and the storage and batching policies. Van Gils et al. (2018c) provided an extensive list of studies that analyze the interaction of these factors on the performance of the chosen routing policy.

Several routing policies are found in the OPP literature. The *return policy* considers that each picker enters and leaves through the same end of aisles containing pick items. In the *S-shape policy*, also known as traversal policy, a picker enters an aisle that contains a product to be picked and traverses it until its end. A return is allowed in the furthest location containing a pick of the last aisle visited, if it is advantageous (Roodbergen and De Koster, 2001). In the *midpoint policy*, pickers are able to go as far as the middle of the aisle before returning and leaving the aisle from where they entered. Exceptions are applied for the first and last aisles containing items so that the picker can access the opposite cross aisle and return to the I/O point after picking all items. In the *largest gap policy*, pickers enter aisles as far as to the point where the item that has the largest gap to another item, or the closest cross aisle, is located. Gaps are calculated as the distance between two adjacent items in the aisle or, if the item is the first or last item in the aisle, the distance between this item and the closest cross aisle. Then, the pickers return and leave at the same end that they entered. As in the midpoint policy, a picker never crosses the entire aisle except for the first and last ones containing items. Other policies include the combined and the aisle-by-aisle (Petersen, 1997; Roodbergen and De Koster, 2001). Examples of each of these routing policies are presented in Figure 2.2, along with the optimal route and its cost in parentheses. Although most policies presented here are for a single-block layout, they are easily adaptable to multi-block cases (Roodbergen and De Koster, 2001).

2.2.3 Discussions on the interactions between SLAP and OPP

When analyzing the SLAP and OPP simultaneously, the main question raised is which OPP policy performs the best for a given SLAP policy, and vice-versa. When deciding the route in the OPP, the product locations must be known prior to solving the problem. Nevertheless, the storage locations may be determined even if a routing strategy is not assumed *a priori*. Studies that analyze the interactions between the two problems consider that the SLAP and



(a) Optimal (76) (b) S-shape (98) (c) Return (102)

(d) Midpoint (90) (e) Largest gap (88)

Figure 2.2 – Routing policies

the OPP are solved in sequence. First, products are allocated to storage locations using a storage policy. Then, different routing policies are tested and their performances are evaluated to determine the overall performance of the system. Table 2.1 presents an overview of the studies that considered multiple policies for the SLAP and the OPP to analyze well-performing combinations under different circumstances.

Table 2.1 – Overview of the papers that investigated multiple storage and routing policies

Reference	*Storage policies*					*Routing policies*	
	Random	*Class-based*		*Dedicated*		*Optimal*	*Heuristic*
		Demand	*Distance*	*Demand*	*Distance*		
Chan and Chan (2011)	✓	COI	Wa/Ot				Ss/Re/Ot
Chen et al. (2010)	✓	Fr	Wa/Ot				Ss/Re/Lg/Ot
Dekker et al. (2004)		Fr	Wa/Ot			✓	Ss/Mp/Lg/Ot
Dijkstra and Roodbergen (2017)		Fr	Di/Wa/Ot				Ss/Re/Mp/Lg
Hsieh and Huang (2011)	✓	Fr	Wa			✓	Ss/Ot
Hsieh and Tsai (2006)		Fr/Af	Di				Ss/Re
Pan and Wu (2012)	✓			Fr	Di/Wa/Aa		Ss/Re/Ot
Petersen (1999)	✓			Fr	Di/Wa	✓	Ss/Lg/Ot
Petersen and Aase (2004)	✓	Fr	Wa	Fr	Wa	✓	Ss/Ot
Petersen et al. (2004)				Fr	Di/Wa/Ot	✓	Ss
Quader and Castillo-Villar (2018)	✓	Fr	Wa/Ot				Ss/Re/Mp/Lg
Roodbergen et al. (2015)	✓	Fr	Di/Wa/Ot				Ss/Re/Lg/Ot
Theys et al. (2010)	✓			Fr	Wa	✓	Ss/Lg/Ot
Van Gils et al. (2018a)	✓	Fr	Wa/Ot				Ss/Re/Mp
Van Gils et al. (2018b)	✓	Fr	Di/Wa/Ot			✓	Ss/Re/Lg/Ot
Van Gils et al. (2019b)	✓	Fr	Di/Wa/Ot			✓	Ss/Re/Mp/Ot

*Fr: Frequency-based, Af: Affinity-based, COI: Cube per order index, Wa: Within-aisle, Di: Diagonal, Ss: S-shape, Re: Return, Mp: Midpoint, Lg: Largest gap, Ot: Other

When demands are known only at the product level, i.e., we have a good approximation of

the number of times each product will be ordered, the best way to solve the SLAP is using a simple storage policy due to the uncertainties involved in the batching of orders to create pick lists. In this case, it is useful to know the interactions between storage and routing policies so that a good combination is used. Otherwise, when demands are known at the order level, i.e., we know not only the demand of each product but also the products contained in each pick list, it is possible to precisely evaluate SLAP solutions by calculating the exact routes that will be performed to retrieve the products in each picking route. This might be the case of markets with stable demands or those in which customers order with more antecedence, such that picking may be performed in waves (Kim and Smith, 2012). This approach is possible even on some unstable markets depending on the availability of data to forecast demands. In such situations, companies may decide to perform reassignment operations to better place the products, for example, in the beginning of the day or the week, in order to minimize picking effort during the period (Kofler et al., 2014).

All cited studies so far consider the decisions on both problems are made sequentially. The integration of location and routing decisions is not new and has proved to be beneficial both in warehousing (Van Gils et al., 2018b,c) and in general logistics problems (Nagy and Salhi, 2007; Salhi and Rand, 1989). If the routing policy is given as an input for the SLAP, and future orders are known or predictable, the storage area setup can be determined optimally for that policy. For instance, Boysen and Stephan (2013) model a SLAP with return routing using dynamic programming considering a layout with only one rack (half-aisle), also proving the NP-hardness of the case with a single aisle. Although some studies explore SLAPs with the assumption of a routing policy, only Mantel et al. (2007) and Dijkstra and Roodbergen (2017) consider a manual picker-to-parts warehouse with demands known at the product or order level, and optimal assignments at single storage locations level in a more complex layout than with a single aisle. They also compare solutions obtained for the optimal SLAP for different routing policies. However, Mantel et al. (2007) assume a specific warehouse system (vertical lift module) and present an MIP for the SLAP with an S-shape routing policy where a return is not allowed in the last aisle. Compared to them, we do not assume any specific warehouse system, and we present a mathematical model for the problem with a slightly different S-shape, besides other routing policies. Dijkstra and Roodbergen (2017) present a mathematical model to generate near-optimal storage location assignments using dynamic programming for the four routing policies presented here. The route lengths are calculated from picking probabilities for product retrieval instead of static pick lists as considered here. Their findings confirm the best combinations of storage and routing policies, e.g., return combines with diagonal, S-shape with within-aisle, and largest gap and midpoint both with within-aisle, among other storage policies. However, Dijkstra and Roodbergen (2017) did not evaluate the SLAP performance for the optimal OPP solution, which is one of the objectives of this study.

2.3 Mathematical models

In this section, we present MIP models to solve the general integrated SLOPP, as well as the four special cases considered here (SL+Re, SL+Ss, SL+Mp and SL+Lg). The general SLOPP model is derived from a TSP one. It is designed to accept any warehouse structure, so it is not required to know specific layout parameters *a priori*, such as the number of aisles, number of slots, etc. After, we adapt it to the special case of a rectangular single-block warehouse layout, which is also used as a reference to model the other four SLOPP special cases. Finally, the linearization technique used for all nonlinear models is briefly described.

2.3.1 General integrated SLOPP

A general formulation for the SLOPP is described next. The term *general* is used to indicate that the problem does not assume any storage layout. This is the most integrated and flexible version of the problem as it does not assume any policy or heuristic underneath. We consider a set $\mathcal{L} = \{1, \ldots, L\}$ of locations representing the slots available, and $\mathcal{L}^* = \mathcal{L} \cup \{0\}$ the set including location 0 representing the I/O point. A set $\mathcal{P} = \{1, \ldots, P\}$ of products to be located is defined, with $\mathcal{P}^* = \mathcal{P} \cup \{0\}$ being the set that includes a dummy product 0 that is assigned to the I/O point. Each product in $\mathcal{P}$ must be assigned to exactly one location in $\mathcal{L}$, thus $P \leq L$. Products not demanded by any order may be ignored from $\mathcal{P}$ without loss of generality of the model. A set $\mathcal{O} = \{1, \ldots, O\}$ of orders is known, and each order o contains a pick list $\mathcal{Q}_o \subset \mathcal{P}^*$, so that the products to be picked in each route are known in advance. Although we consider here a pick-by-order environment, where pick lists are composed solely of products from a single order, $\mathcal{Q}_o$ can also represent a previously defined batch of different customer orders that will be picked together without loss of generality. Since all routes start and end in the I/O point, the product 0 is contained in all pick lists. The matrix $\mathbf{D} = \{d_{ij}\}$ represents the cost (distance, time, etc.) for a picker to move from location i to location j. Note that values in $\mathbf{D}$ can be set to zero to represent products located in the same location. The binary routing variable x_{ij}^o is equal to one if the route to pick the items in the o-th order contains the path going from location i to location j, or zero otherwise. We assume that there is enough capacity for the picker to retrieve all products from o in only one route. The assignment variable y_{ki} is also binary and equals to one if product k is assigned to location i. $y_{00} = 1$ is defined to locate the I/O point at location 0. Finally, we define auxiliary variables u_k^o, which are used to create a valid route for order o, representing in which position product $k \in \mathcal{Q}_o \setminus \{0\}$ is retrieved in the route. The general SLOPP is formulated as follows:

$$\text{Min} \sum_{o \in \mathcal{O}} \sum_{k \in \mathcal{Q}_o} \sum_{l \in \mathcal{Q}_o \setminus \{k\}} \sum_{i \in \mathcal{L}^*} \sum_{j \in \mathcal{L}^* \setminus \{i\}} d_{ij} x_{ij}^o y_{ki} y_{lj}, \tag{2.1}$$

subject to

$$\sum_{i \in \mathcal{L}} y_{ki} = 1, \quad \forall k \in \mathcal{P}, \tag{2.2}$$

$$\sum_{k \in \mathcal{P}} y_{ki} \leq 1, \quad \forall i \in \mathcal{L}, \tag{2.3}$$

$$\sum_{k \in \mathcal{Q}_o} y_{ki} = \sum_{j \in \mathcal{L}^* \setminus \{i\}} x^o_{ij}, \quad \forall o \in \mathcal{O}, i \in \mathcal{L}^*, \tag{2.4}$$

$$\sum_{k \in \mathcal{Q}_o} y_{kj} = \sum_{i \in \mathcal{L}^* \setminus \{j\}} x^o_{ij}, \quad \forall o \in \mathcal{O}, j \in \mathcal{L}^*, \tag{2.5}$$

$$u^o_k - u^o_l + |\mathcal{Q}_o| x^o_{ij} y_{ki} y_{lj} \leq |\mathcal{Q}_o| - 1, \quad \forall o \in \mathcal{O}, k, l \in \mathcal{Q}_o \setminus \{0\}, k \neq l, i, j \in \mathcal{L}^*, i \neq j, \tag{2.6}$$

$$0 \leq u^o_k \leq |\mathcal{Q}_o| - 1, \quad \forall k \in \mathcal{Q}_o \setminus \{0\}, o \in \mathcal{O}, \tag{2.7}$$

$$x^o_{ij} \in \{0, 1\}, \quad \forall o \in \mathcal{O}, i, j \in \mathcal{L}^*, i \neq j, \tag{2.8}$$

$$y_{ki} \in \{0, 1\}, \quad \forall k \in \mathcal{P}, i \in \mathcal{L}^*. \tag{2.9}$$

The objective function (2.1) minimizes the total routing cost. Constraints (2.2) assign each product to exactly one location. Constraints (2.3) define that each location can have no more than one product assigned to it. Constraints (2.4) and (2.5) link the assignment and routing variables. They ensure that when there is a product k in order o located in a certain slot, then one path must be chosen that leaves and enters this slot. Constraints (2.6) eliminate subtours in the routes by guaranteeing that if product k is located in i, product l is located in j, and slot j is visited after slot i in the route, then $u^o_l > u^o_k$, where l and k are products in the pick list. These constraints are derived from the Miller-Tucker-Zemlin (MTZ) formulation for the TSP as presented in Scholz et al. (2016) for the OPP. Constraints (2.7) define the bound of the auxiliary variables, while constraints (2.8) and (2.9) define the domain of the routing and assignment variables, respectively.

We now describe the specific warehouse layout case examined in this paper. We consider a picker-to-product system for a rectangular warehouse with a single block. This is a common layout studied in literature (Dijkstra and Roodbergen, 2017; Petersen and Aase, 2004; Roodbergen and De Koster, 2001). The storage area has a set $\mathcal{A} = \{1, \ldots, A\}$ of parallel aisles with a set $\mathcal{B} = \{1, \ldots, B\}$ of rows of slots. Each row, including those in the first and last aisles, contains two slots, i.e., one in the left rack and another in the right rack. We ignore the distance between the two slots in the same row (two-sided picking). We denote (a, b) as the slot located in aisle a and row b. Due to the two-sided picking, we can ignore if it is on the left or right side of the row. A cross aisle exists both in the front and the back of the aisles. The distance a picker has to move from the entrance of an aisle to the entrance of an adjacent aisle is equal to M. Each slot has the capacity for a product with a sufficient number of single items meaning that no replenishment is needed during the picking. All products have the same physical requirements (space, temperature, etc.). Two neighbor rows within the same aisle

have distance N, which is also the distance a picker has to travel to move from the head of the aisle in any cross aisle to the first row in the aisle. Pickers may perform a return anywhere at no cost, which is a common assumption when the picking equipment allows back and forth movement change with no additional effort. The effects of congestion is negligible, i.e., time to remove items is small enough to avoid pick-column blocking, and aisles allow picker passing each other, avoiding in-the-aisle blocking. Finally, an I/O point is located at the head of the first aisle. We illustrate this setting in Figure 2.3. It represents the floor plan of a warehouse with $A = 5$ aisles and $B = 5$ rows in each aisle.

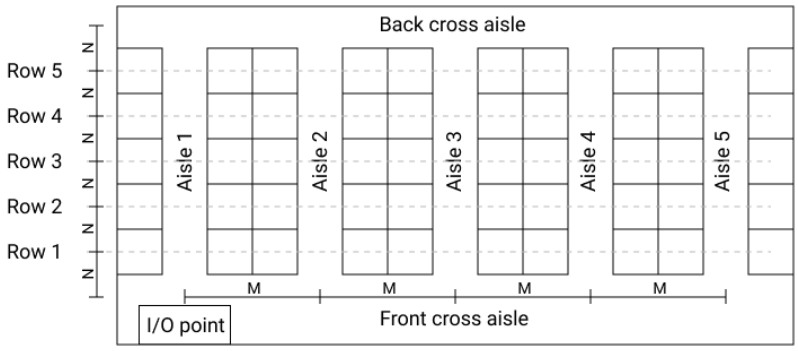


Figure 2.3 – Representation of a warehouse with two-sided parallel aisles in a single block

In order to solve the SLOPP for the single-block warehouse, the cost matrix $\mathbf{D}$ is determined by solving a special case of the shortest path problem between every pair of slots. The minimum distance d_{ij} between two slots located in $i = (a_i, b_i)$ and $j = (a_j, b_j)$ is calculated as:

$$d_{ij} = M(|a_i - a_j|) + \begin{cases} N|b_i - b_j|, & \text{if } a_i = a_j, \\ N \min\{(b_i + b_j), (2B + 2 - b_i - b_j)\}, & \text{otherwise.} \end{cases} \tag{2.10}$$

The first term gives the distance traveled in the cross aisles, while the second term gives the distance within-aisle either when both slots are in the same aisle or when they are in different aisles. The *min* function represents the decision of using the cross aisle in the front or in the back of the storage area to change aisles. The SLOPP may now be solved using the distance matrix as defined.

2.3.2 Storage location with different routing policies

It is also possible to model reductions of the SLOPP for the special case of a single-block warehouse when considering any routing policy other than the optimal one, which we refer to as *SL+routing policy*.

Consider D_o^{CA} as being the distance traveled by the picker in the cross aisles and D_o^{WA} as the distance traveled within the aisles to pick products from order o. For any routing strategy,

the objective of the problem is to minimize the total distance traveled to fulfill all required orders given as

$$\min \sum_{o \in \mathcal{O}} \left(D_o^{CA} + D_o^{WA} \right). \tag{2.11}$$

In order to calculate D_o^{CA} and D_o^{WA}, we should define the locations of the products to be picked in each route. Considering the set of products ($\mathcal{P}$) to be located, and the single-block layout as described in Section 2.3.1, we define the binary location variable y_{ab}^k as equal to one if product $k \in \mathcal{P}$ is assigned to the slot located in (a, b), and zero otherwise. Thus, the following constraints can be used to assign all products in $\mathcal{P}$ to locations (constraints (2.12)) such that no location has more than two products (2.13), one on each side. Thus, D_o^{CA} and D_o^{WA} are functions dependent of y.

$$\sum_{a \in \mathcal{A}} \sum_{b \in \mathcal{B}} y_{ab}^k = 1, \quad \forall k \in \mathcal{P}, \tag{2.12}$$

$$\sum_{k \in \mathcal{P}} y_{ab}^k \leq 2, \quad \forall a \in \mathcal{A}, b \in \mathcal{B}, \tag{2.13}$$

$$y_{ab}^k \in \{0, 1\}, \quad \forall k \in \mathcal{P}, a \in \mathcal{A}, b \in \mathcal{B}. \tag{2.14}$$

For the sake of easiness to understand the models, we define new auxiliary variables to compute D_o^{CA} and D_o^{WA} for different routing policies instead of representing them by a function of y. We summarize in Table 2.2 the new auxiliary variables used in the four models we designed to solve the storage location (SL) with return (SL+Re), S-shape (SL+Ss), midpoint (SL+Mp), and largest gap (SL+Lg) routing policies, also indicating in which models they are used.

Table 2.2 – Notation of the models

Variable	Description	Re	Ss	Mp	Lg
f_{oa}	Furthest row containing a pick in aisle a	✓	✓	✓	✓
z_{oa}	If there is a pick in aisle a	✓	✓	✓	✓
v_o	Last aisle containing a pick	✓		✓	✓
v_{oa}	If aisle a is the last aisle containing a pick		✓	✓	✓
k_o	Auxiliary variable used to calculate s_o^{SS}		✓		
s_o^{SS}	(S-shape special case) If the number of aisles with picks is odd		✓		
f_{oa}^1	Furthest row from lower cross aisle below midpoint containing a pick in aisle a			✓	
f_{oa}^2	Furthest row from upper cross aisle above midpoint containing a pick in aisle a			✓	
z_{oa}^1	If there is a pick in a row below midpoint in aisle a			✓	
z_{oa}^2	If there is a pick in a row above midpoint in aisle a			✓	
u_o^2	First aisle containing a pick above midpoint			✓	
u_{oa}^2	If aisle a is the first aisle containing a pick above midpoint			✓	
s_o^{MP}	(Midpoint special case) If $u_o^2 \neq 0$ and $u_o^2 \neq v_o$			✓	
w_{oa}^{MP}	If aisle a is neither the first aisle above midpoint nor the last aisle with a pick			✓	
G_{oa}	Largest gap between two neighbor picks or a pick and a neighbor cross aisle in aisle a				✓
g_{oab}	Gap between a pick in row $b \in B \cup \{0\}$ to nearest pick in another row or cross aisle in aisle a. When $b = 0$ we have the largest gap from the lower cross aisle to the first pick				✓
h_{oab}	Auxiliary variable used to calculate the largest gap				✓
u_o	First aisle containing a pick				✓
u_{oa}	If aisle a is the first one containing a pick				✓
s_o^{LG}	(Largest gap special case) If $u_o \neq v_o$				✓
w_{oa}^{LG}	If aisle a is between the first and last aisles with a pick				✓

Storage location with return routing (SL+Re)

Considering the sets, variables, and parameters as previously defined, we represent the distances traveled in the SL+Re as follows. For any route $o \in \mathcal{O}$, $D_o^{CA} = 2M(v_o - 1)$, i.e., the distance to travel back and forth between the I/O point and the last aisle containing a product to be picked. Furthermore, the distance traveled within all aisles is $D_o^{WA} = 2N \sum_{a \in A} f_{oa}$, i.e., the sum of the distances to go back and forth from the front cross aisle to the furthest location that contains a pick in each aisle. In order to calculate v_o and f_{oa}, we added the following constraints to the base model (2.11)–(2.14) to create a model for the SL+Re:

$$f_{oa} \geq \sum_{b \in \mathcal{B}} b y_{ab}^k, \quad \forall o \in \mathcal{O}, a \in \mathcal{A}, k \in \mathcal{Q}_o \setminus \{0\}, \tag{2.15}$$

$$f_{oa} \leq B z_{oa}, \quad \forall o \in \mathcal{O}, a \in \mathcal{A}, \tag{2.16}$$

$$\textbf{if } z_{oa} = 1 \textbf{ and } \sum_{c=(a+1)}^{A} z_{oc} = 0 \rightarrow v_o = a, \quad \forall o \in \mathcal{O}, a \in \mathcal{A}, \tag{2.17}$$

$$z_{oa} \in \{0, 1\}, \quad \forall o \in \mathcal{O}, a \in \mathcal{A}, \tag{2.18}$$

$$0 \leq f_{oa} \leq B, \quad \forall o \in \mathcal{O}, a \in \mathcal{A}, \tag{2.19}$$

$$1 \leq v_o \leq A, \quad \forall o \in \mathcal{O}. \tag{2.20}$$

Constraints (2.15) define a lower bound for f_{oa}. Since we minimize f_{oa} in the objective function, there is no need to limit its upper bound. Constraints (2.16) set z_{oa} to one if f_{oa} assumes a value greater than zero. z_{oa} is used to determine if a is the last aisle containing a product from order o, then $v_o = a$. These constraints can be written using indicator (or logical) constraints available in modern MIP solvers as presented in constraints (2.17). The remaining constraints (2.18)–(2.20) define the domain of the decision variables.

Storage location with S-shape routing (SL+Ss)

For the SL+Ss, the distances traveled are calculated as follows. The distance in the cross aisles is $D_o^{CA} = \sum_{a \in \mathcal{A}} 2M v_{oa}(a - 1)$, which is similar to the return routing, but using the equivalent binary decision variables v. The use of a binary v is more adequate to calculate the distance within-aisles given as $D_o^{WA} = \sum_{a \in \mathcal{A}} (N(B + 1)z_{oa} + 2N s_o^{SS} f_{oa} v_{oa}) - N(B + 1)s_o^{SS}$. The expression $N(B + 1)$ represents the distance traveled when the entire aisle is crossed by the picker. So, according to the first term of the equation, when there are products to be picked in an aisle, the picker crosses it entirely. Since this term also considers the last aisle independently of the value of s_o^{SS}, we have to add the last two terms to correct the distance traveled in the last aisle when the picker performs a return in it. Thus, the SL+Ss is modeled using the base model (2.11)–(2.14), constraints (2.15), (2.16), (2.18) and (2.19) to calculate

z_{oa} and f_{oa}, and the following constraints to determine v_{oa} and s_o^{SS}:

$$\textbf{if } z_{oa} = 1 \textbf{ and } \sum_{c=(a+1)}^{A} z_{oc} = 0 \rightarrow v_{oa} = 1, \quad \forall o \in \mathcal{O}, a \in \mathcal{A}, \tag{2.21}$$

$$\sum_{a \in \mathcal{A}} z_{oa} = 2k_o + s_o^{SS}, \quad \forall o \in \mathcal{O} \tag{2.22}$$

$$v_{oa} \in \{0,1\}, \quad \forall o \in \mathcal{O}, a \in \mathcal{A}, \tag{2.23}$$

$$s_o^{SS} \in \{0,1\}, \quad \forall o \in \mathcal{O}, \tag{2.24}$$

$$0 \leq k_o \leq A/2, \quad \forall o \in \mathcal{O}. \tag{2.25}$$

Constraints (2.21) are the equivalent indicator constraints to set the appropriate value for the binary variable v. The variable s_o^{SS} is calculated as the remainder of $\frac{\sum_{a \in \mathcal{A}} z_{oa}}{2}$ using constraints (2.22). Constraints (2.23)–(2.25) define the domain of the variables v_{oa}, s_o^{SS} and k_o, respectively. We note that our model is rather different than that of Mantel et al. (2007) because we consider the return option at the last aisle, which makes the problem significantly harder to solve.

Storage location with midpoint routing (SL+Mp)

To the formulation of the SL+Mp, we have divided the storage area into two zones. Midpoint is defined as the row $p = \lceil B/2 \rceil$. Zone 1 ($\mathcal{B}^1 = \{1, \ldots, p\}$) contains all locations below p, including p itself. Meanwhile, zone 2 ($\mathcal{B}^2 = \{p+1, \ldots, B\}$) contains all locations above p. This definition is used to create variables z^1 and z^2, which are the z equivalent of each zone, as well as f^1 and f^2, which are their f equivalents. These, along with other exclusive auxiliary variables for this routing policy, are used to calculate the distances traveled as follows. As in the return routing, $D_o^{CA} = 2M(v_o - 1)$. Furthermore, $D_o^{WA} = 2N((B+1)s_o^{MP} + \sum_{a \in A}(f_{oa}^1 w_{oa}^{MP} + f_{oa}^2 w_{oa}^{MP} + f_{oa} v_{oa} - f_{oa} v_{oa} s_o^{MP}))$. This long equation can be explained by dividing it into three parts. The first part ($2N(B+1)s_o^{MP}$) is the distance traveled to cross the first and last aisles entirely when s_o^{MP} is activated, i.e., there is at least one aisle with a product to be picked above the midpoint and it is different than the last aisle with a product to be picked. The second part ($\sum_{a \in A}(f_{oa}^1 w_{oa}^{MP} + f_{oa}^2 w_{oa}^{MP})$) is the distance traveled to the rows closest to the midpoint in all aisles except for the first and the last. The third part ($\sum_{a \in A}(f_{oa} v_{oa} - f_{oa} v_{oa} s_o^{MP})$) models the special case when there is only one aisle with products to be picked above midpoint and it is the last aisle. In this case, we ignore the zones and calculate the furthest row containing a product to be picked in the last aisle using variable f to determine the return point within the aisle. The SL+Mp is modeled using the base model (2.11)–(2.14), constraints (2.15) and (2.17) to calculate f_{oa} and v_o, and the following constraints to determine f_{oa}^1, f_{oa}^2, z_{oa}^1, z_{oa}^2, z_{oa}, u_o^2, u_{oa}^2, v_{oa}, s_o^{MP}, and w_{oa}^{MP}:

$$\textbf{if } \sum_{b \in \mathcal{B}^r} \sum_{k \in \mathcal{Q}_o} y_{ab}^k = 0 \rightarrow f_{oa}^r = 0, \quad o \in \mathcal{O}, a \in \mathcal{A}, r = \{1,2\}, \tag{2.26}$$

$$\textbf{if } \sum_{k\in\mathcal{Q}_o} y^k_{ab} \geq 1 \textbf{ and } \sum_{d=(b+1)}^{p} \sum_{k\in\mathcal{Q}_o} y^k_{ad} = 0 \rightarrow f^1_{oa} = b, \quad o \in \mathcal{O}, a \in \mathcal{A}, b \in \mathcal{B}^1 \tag{2.27}$$

$$\textbf{if } \sum_{k\in\mathcal{Q}_o} y^k_{ab} \geq 1 \textbf{ and } \sum_{d=p+1}^{b-1} \sum_{k\in\mathcal{Q}_o} y^k_{ad} = 0 \rightarrow f^2_{oa} = (B+1) - b, \quad o \in \mathcal{O}, a \in \mathcal{A}, b \in \mathcal{B}^2 \tag{2.28}$$

$$\textbf{if } f^r_{oa} = 0 \rightarrow z^r_{oa} = 0, \quad o \in \mathcal{O}, a \in \mathcal{A}, r = \{1,2\}, \tag{2.29}$$

$$f^r_{oa} \leq B z^r_{oa}, \quad o \in \mathcal{O}, a \in \mathcal{A}, r = \{1,2\}, \tag{2.30}$$

$$z_{oa} \geq z^r_{oa}, \quad o \in \mathcal{O}, a \in \mathcal{A}, r = \{1,2\}, \tag{2.31}$$

$$z_{oa} \leq \sum_{r=1}^{2} z^r_{oa}, \quad o \in \mathcal{O}, a \in \mathcal{A}, \tag{2.32}$$

$$\textbf{if } \sum_{a\in\mathcal{A}} z^2_{oa} = 0 \rightarrow u^2_o = 0, \quad o \in \mathcal{O}, \tag{2.33}$$

$$\textbf{if } z^2_{oa} = 1 \textbf{ and } \sum_{c=1}^{a-1} z^2_{oc} = 0 \rightarrow u^2_o = a, \quad o \in \mathcal{O}, a \in \mathcal{A}, \tag{2.34}$$

$$\textbf{if } u^2_o \neq 0 \textbf{ and } u^2_o \neq v_o \rightarrow s^{MP}_o = 1, \quad o \in \mathcal{O}, \tag{2.35}$$

$$\textbf{if } u^2_o = 0 \textbf{ or } u^2_o = v_o \rightarrow s^{MP}_o = 0, \quad o \in \mathcal{O}, \tag{2.36}$$

$$u^2_o = \sum_{a\in\mathcal{A}} a u^2_{oa}, \quad o \in \mathcal{O}, \tag{2.37}$$

$$\sum_{a\in\mathcal{A}} u^2_{oa} \leq 1, \quad o \in \mathcal{O}, \tag{2.38}$$

$$v_o = \sum_{a\in\mathcal{A}} a v_{oa}, \quad o \in \mathcal{O}, \tag{2.39}$$

$$\sum_{a\in\mathcal{A}} v_{oa} \leq 1, \quad o \in \mathcal{O}, \tag{2.40}$$

$$w^{MP}_{oa} \leq 1 - u^2_{oa}, \quad o \in \mathcal{O}, a \in \mathcal{A}, \tag{2.41}$$

$$w^{MP}_{oa} \leq 1 - v_{oa}, \quad o \in \mathcal{O}, a \in \mathcal{A}, \tag{2.42}$$

$$w^{MP}_{oa} \geq 1 - u^2_{oa} - v_{oa}, \quad o \in \mathcal{O}, a \in \mathcal{A}, \tag{2.43}$$

$$0 \leq f^1_{oa}, f^2_{oa} \leq p, \quad \forall o \in \mathcal{O}, a \in \mathcal{A}, \tag{2.44}$$

$$z^1_{oa}, z^2_{oa}, z_{oa}, u^2_{oa}, v_{oa}, s^{MP}_o, w^{MP}_{oa} \in \{0,1\}, \quad \forall o \in \mathcal{O}, a \in \mathcal{A}, \tag{2.45}$$

$$0 \leq u^2_o \leq A, \quad \forall o \in \mathcal{O}. \tag{2.46}$$

Constraints (2.26)–(2.28) determine the furthest row to be traveled in each zone of each aisle. Constraints (2.29) and (2.30) set the proper values to variables z^1 and z^2. Constraints (2.31) and (2.32) set the proper value to variables z. Constraints (2.33) and (2.34) determine the first aisle with products to be picked above the midpoint. Constraints (2.35) and (2.36) set s^{MP} to its appropriate value. Constraints (2.37) set proper values to the integer and binary versions of u^2 and constraints (2.38) certify that no more than one binary is activated. Constraints

(2.39) and (2.40) do the same for v. Constraints (2.41)–(2.43) set w^{MP} to its correct value. Finally, constraints (2.44)–(2.46) define the domain of the variables.

Storage location with largest gap routing (SL+Lg)

The SL+Lg is modeled as follows. The distance traveled in the cross aisles is similar to the return and midpoint, given as $D_o^{CA} = 2M(v_o - 1)$. To calculate the distance traveled within aisles, like in the SL+Mp, we have to consider the different cases when only the lower cross aisle or both cross aisles are used. The first case is obeserved when all picked items are located in a single aisle. If more than two aisles should be entered, then the first and last are entirely crossed, while in the intermediary ones returns are performed to avoid the distance from the largest gap. Thus, the distance within-aisle is given as $D_o^{WA} = 2N((B + 1)s_o^{LG} + \sum_{a\in A}(s_o^{LG}w_{oa}^{LG}(B + 1 - G_{oa}) + (1 - s_o^{LG})f_{oa}v_{oa}))$. The first term in the formula $((B + 1)s_o^{LG})$ represents the first and last aisles crossed entirely when s^{LG} is active. The second term $(\sum_{a\in A}(s_o^{LG}w_{oa}^{LG}(B + 1 - G_{oa})))$ is the distance traveled in the intermediary aisles given by the total number of rows minus the largest gap in that aisle. The third term $(\sum_{a\in A}((1 - s_o^{LG})f_{oa}v_{oa}))$ is the special case when all picks are in the same aisle, so that a simple return at the furthest row containing a pick is performed. Now, we model the SL+Lg using the base model (2.11)–(2.14), constraints (2.15)–(2.17), (2.39), (2.40) to calculate f_{oa}, z_{oa}, v_o, v_{oa}, and the following constraints to determine u_o, u_{oa}, g_{oab}, G_{oa}, w_{oa}^{LG}, and h_{oab}:

$$\textbf{if } z_{oa} = 1 \textbf{ and } \sum_{c=1}^{a-1} z_{oc} = 0 \rightarrow u_o = a, \quad o \in \mathcal{O}, a \in \mathcal{A}, \tag{2.47}$$

$$u_o = \sum_{a\in\mathcal{A}} a u_{oa}, \quad o \in \mathcal{O}, \tag{2.48}$$

$$\sum_{a\in\mathcal{A}} u_{oa} = 1, \quad o \in \mathcal{O}, \tag{2.49}$$

$$\textbf{if } u_o \leq (a-1) \textbf{ and } v_o \geq (a+1) \rightarrow w_{oa}^{LG} = 1, \quad o \in \mathcal{O}, a \in \mathcal{A}, \tag{2.50}$$

$$\textbf{if } u_o \geq a \textbf{ or } v_o \leq a \rightarrow w_{oa}^{LG} = 0, \quad o \in \mathcal{O}, a \in \mathcal{A}, \tag{2.51}$$

$$\textbf{if } u_o = v_o \rightarrow s_o^{LG} = 0, \quad o \in \mathcal{O}, \tag{2.52}$$

$$\textbf{if } u_o \neq v_o \rightarrow s_o^{LG} = 1, \quad o \in \mathcal{O}, \tag{2.53}$$

$$\textbf{if } z_{oa} = 0 \rightarrow g_{oa0} = 0, \quad o \in \mathcal{O}, a \in \mathcal{A}, \tag{2.54}$$

$$\textbf{if } \sum_{k\in\mathcal{Q}_o} y_{oab}^k \geq 1 \textbf{ and } \sum_{d=1}^{b-1} \sum_{k\in\mathcal{Q}_o} y_{oad}^k = 0 \rightarrow g_{oa0} = b, \quad o \in \mathcal{O}, a \in \mathcal{A}, b \in \mathcal{B}, \tag{2.55}$$

$$\textbf{if } \sum_{k\in\mathcal{Q}_o} y_{oab}^k = 0 \rightarrow g_{oab} = 0, \quad o \in \mathcal{O}, a \in \mathcal{A}, b \in \mathcal{B}, \tag{2.56}$$

$$\textbf{if } \sum_{k\in\mathcal{Q}_o} y^k_{oab_1} \geq 1 \textbf{ and } \sum_{k\in\mathcal{Q}_o} y^k_{oab_2} \geq 1 \textbf{ and } \sum_{b_3=(b_1+1)}^{b_2-1} \sum_{k\in\mathcal{Q}_o} y^k_{oab_3} = 0 \rightarrow g_{oab_1} = b_2 - b_1, \quad o \in \mathcal{O}, a \in \mathcal{A}, b_1, b_2 \in \mathcal{B}, \tag{2.57}$$

$$\textbf{if } \sum_{k\in\mathcal{Q}_o} y^k_{oab_1} \geq 1 \textbf{ and } \sum_{b_2=(b_1+1)}^{B} \sum_{k\in\mathcal{Q}_o} y^k_{oab_2} = 0 \rightarrow g_{oab_1} = B + 1 - b_1, \quad o \in \mathcal{O}, a \in \mathcal{A}, b_1 \in \mathcal{B}, \tag{2.58}$$

$$G_{oa} \geq g_{oab}, \quad o \in \mathcal{O}, a \in \mathcal{A}, b \in \mathcal{B}, \tag{2.59}$$

$$G_{oa} \leq g_{oab} + \sum_{b_2\in B\cup\{0\}} (B+1)h_{oab_2}, \quad o \in \mathcal{O}, a \in \mathcal{A}, b \in \mathcal{B}, \tag{2.60}$$

$$\sum_{b\in\mathcal{B}\cup\{0\}} h_{oab} = 1, \quad o \in \mathcal{O}, a \in \mathcal{A}, \tag{2.61}$$

$$u_o, g_{oab}, G_{oa} \geq 0, \quad \forall o \in \mathcal{O}, a \in \mathcal{A}, \tag{2.62}$$

$$u_{oa}, w^{LG}_{oa}, h_{oab} \in \{0,1\}, \quad \forall o \in \mathcal{O}, a \in \mathcal{A}, b \in \mathcal{B}. \tag{2.63}$$

Constraints (2.47) determine the first aisle with a pick. Constraints (2.48) and (2.49) set proper values for u_o and u_{oa}. Constraints (2.50) and (2.51) use indicator constraints to ensure that $w^{LG}_{oa} = 1$ when $u_o < a < v_o$, and zero otherwise. Constraints (2.52) and (2.53) guarantee that $s^{LG} = 1$ only if the first and last aisles are different. If there is no pick in the aisle, then the gap is set to zero in constraints (2.54). Constraints (2.55) make sure that if there is a product in row b and there is no product between the lower cross aisle and row $b-1$, then the gap from the lower cross aisle to the first pick is b. Constraints (2.56) set the gap in row b to zero when there is no pick in b. Constraints (2.57) state that if there is a pick in b_1 and b_2, and no other pick between the two, then the gap in row b_1 is equal to $b_2 - b_1$. Constraints (2.58) handle the special case when the pick in b_1 is the last pick in the aisle. So, the gap in b_1 is the distance from it to the upper cross aisle. Constraints (2.59)–(2.61) are linear forms to compute the largest value between all gaps, i.e., $G_{oa} = \max(g_{oab} | b \in \mathcal{B} \cup \{0\})$. Finally, constraints (2.62) and (2.63) define the domain of the new variables.

2.3.3 Linearizations

The introduced models, except for the SL+Re, are nonlinear MIP formulations due to the product of variables either in the objective function or in constraints, as in (2.6). Nonlinear MIP models can be reformulated to an equivalent linear representation using a linearization technique. In order to solve the problems, we reformulated the models using the standard linearization technique (Glover and Woolsey, 1974). The technique is used to replace the product of variables as follows:

- product of two binary variables a and b is treated by replacing ab by a new variable c

and adding the constraints $c \leq a$, $c \leq b$ and $c \geq a + b - 1$;

- product of three binary variables a, b, and c, abc is replaced by d and the constraints $d \leq a$, $d \leq b$, $d \leq c$ and $d \geq a + b + c - 2$;
- product of binary variable a and non-binary variable b, ab is replaced by c by adding the constraints $c \leq \overline{b}a$, where $\overline{b}$ is the upper bound value b can assume, $c \leq b$, and $c \geq b - \overline{b}(1 - a)$.

The advantage of linearizing a problem is that it can be solved by any integer linear programming solver. While the standard linearization benefits from its simplicity, it has the disadvantage of requiring a large number of auxiliary variables and constraints to replace the nonlinear terms.

2.4 A General Variable Neighborhood Search for the SLOPP and its special cases

Variable Neighborhood Search (VNS) is a metaheuristic that uses a systematic change of neighborhood within a descent phase to find a local optimum starting from different points generated by a perturbation phase (Mladenović and Hansen, 1997). Among its several variants, the one with some of the most successful applications is the General VNS (GVNS) (Hansen et al., 2019). In the GVNS, the descent phase consists in the application of several local searches (LS) performed in a deterministic way by a method called Variable Neighborhood Descent (VND). The reasoning of VND is that a local optimum in a neighborhood is not necessarily a local optimum in another, such that the use of several LSs are more likely to reach global optimum (Hansen et al., 2019). The implemented VND is shown in Algorithm 1.

We first define an initial solution, as described in Section 2.4.1. Then, a set $\mathcal{N}_l$ of neighborhoods is defined with $l = \{1, 2, 3\}$ representing the three LSs, presented in Section 2.4.2, used for the exploration, and their exploration order is known *a priori* (line 1). The first improving solution found in a neighborhood replaces the current one (*first improvement* strategy) until there is no possibility of improvement in that neighborhood (lines 4–6). Preliminary experiments showed that the first improvement strategy performed better than searching the whole neighborhood for the best improving solution which is too time consuming. The search returns to the first neighborhood when the solution is improved in the current neighborhood. Otherwise, it continues in the next neighborhood (lines 7–11). The solution returned by the VND (line 13) should be a local minimum with respect to all three neighborhoods.

The perturbation phase of the GVNS consists in applying a function *Shake* as presented in Algorithm 2. This function is used to move from solution x to a random solution in the *Slot Exchange* neighborhood. Preliminary experiments showed that the *Slot Exchange*

Algorithm 1 Function VND(x)

1: Neighborhoods $\mathcal{N}_l(x)$, $l = \{1, 2, 3\}$;
2: **repeat**
3: $x_l \leftarrow x$;
4: **while** there is a $x' \in \mathcal{N}_l(x) | f(x') < f(x)$ **do**
5: $x \leftarrow x'$;
6: **end while**
7: **if** $f(x) < f(x_l)$ **then**
8: $l = 1$;
9: **else**
10: $l = l + 1$;
11: **end if**
12: **until** $l < 4$
13: **return** x.

neighborhood performs particularly well. In summary, each shake consists in selecting two random and different products and swap their locations in the storage area (more details about this neighborhood is presented in Section 2.4.2). A total of S swaps is performed before *Shake* ends. Note that small values of S tend to lead to the same local optimum when VND is applied. Otherwise, if S is too high, the new solution may lose the good characteristics of the initial solution, thus behaving like a randomly generated solution.

Algorithm 2 Function Shake(x, S)

1: **for** $s = 1$ **to** S **do**
2: Randomly select $i, j \in \mathcal{P} | i \neq j$;
3: $x \leftarrow$ *SlotExchange*(x, i, j);
4: **end for**
5: **return** x.

The steps of our GVNS are presented in Algorithm 3. Besides an initial solution x, it receives the maximum number of cycles K and shakes in each cycle S, and the time limit T. We start the GVNS by applying the VND to x (line 2). The stopping criteria are either when K or T is reached. In each cycle, we iterate a counter s until S is reached. For each s, a shake followed by a VND is performed (lines 5–6). If the objective function of the new local optimum improves the previous best known solution (BKS), s is reset, and the BKS is updated. Otherwise, s is incremented (lines 7–12). The shake counter is reset at the end of each cycle (line 15).

2.4.1 Generating an initial solution

A feasible solution for the SLOPP may be generated using any combination of a storage and a routing policy, such as those presented in Section 2.2.1.

The storage policy is used to assign products in $\mathcal{P}$ to locations in $\mathcal{L}$, i.e., to an aisle a and row b in the single-block layout. The policies implemented are: (i) random, (ii) dedicated with frequency-based and diagonal, and (iii) dedicated with frequency-based and within-aisle. Given a product $k \in \mathcal{P}$, its frequency is calculated by the number of times that $k \in \mathcal{Q}_o$ for all $o \in \mathcal{O}$. In the diagonal policy, the distance between a slot located in $l = (a, b)$ and the

Algorithm 3 Function GVNS (x, K, S, T)

```
1: k = 1, s = 1, t = 0;
2: x' ← VND(x);
3: while k ≤ K and t ≤ T do
4:    while s ≤ S do
5:       x' ← Shake(x', s);
6:       x' ← VND(x');
7:       if f(x') < f(x) then
8:          x ← x';
9:          s = 1;
10:      else
11:         s = s + 1;
12:      end if
13:   end while
14:   k = k + 1;
15:   s = 1;
16:   t ← Run time;
17: end while
18: return x.
```

I/O point is given by $d_{l0} = M(a-1) + Nb$. Then, the product with the highest frequency is assigned to the closest slot to the I/O point, and so forth. Ties are broken randomly. For the within-aisle policy, the distance between a to the I/O point is calculated as $M(a-1)$. The aisles are sorted from the closest to the farthest, and the closest aisle is filled with the products with the highest frequencies, and so on.

When the products locations are known, the SLOPP reduces to an OPP for each order. Each OPP can be solved using an optimal or a heuristic method. The objective function value of the solution is the sum of the distances traveled by the picker in all routes generated for the OPPs. In the storage location problem with heuristic routing policies, i.e., SL+Re, SL+Ss, SL+Mp and SL+Lg, the computation of the routes is straightforward, and done by verifying where picks are located in each aisle. We highlight that the largest gap requires additional operations to solve the maximum gap problem (Preparata and Shamos, 2012), which can be solved in $O(n)$ using the pigeonhole principle for n points of interest in the aisle, i.e., the items to be retrieved in it and the two end aisles. In the case of the SLOPP, the objective function is obtained by solving an instance of the travelling salesman problem (TSP) for each route, either using an optimal or heuristic method, such as LKH.

2.4.2 Local searches

We defined three neighborhoods to search for local improvements during VND. They are all based on the swap of products locations in the warehouse. Whenever the storage setup is changed, a new call for the routing policy is made to reevaluate the picking routes. A picking route is reevaluated only if a product contained in its pick list is involved in the change, and if the exchanged products do not belong to the same order, which would result in the same route.

The first neighborhood is defined as the swap of products assigned to different slots. The *slot exchange* LS finds two slots with an ordered product located in at least one of them and swaps their assignments. Mathematically, given a slot located in (a_k, b_k) containing a product k, i.e., $y^k_{a_k b_k} = 1$, and a slot located in (a_l, b_l) containing a product l, i.e., $y^l_{a_l b_l} = 1$, the slot exchange reassigns k to the slot in (a_l, b_l), i.e., $y^k_{a_l b_l} = 1$, and l to the slot in (a_k, b_k), i.e., $y^l_{a_k b_k} = 1$. All combinations of pairs of slots are tested except when $a_k = a_l$ and $b_k = b_l$. Since only slots containing products and in different locations are swapped, this neighborhood size is $O(|\mathcal{P}||\mathcal{L}|)$ in the worst case, given that the number of locations $|\mathcal{L}| = AB$. A neighbor is an improving solution either if its objective function is better or, specially for this LS, when it has equal value, but the swap moves a product with higher frequency closer to the I/O point. This second condition proved to be very effective in avoiding bad local optimum.

The second neighborhood is defined as the swap of products contained in different rows. The *row exchange* LS finds two rows with at least one ordered product in at least one of them, even if they are in different aisles, and swaps their products. In the storage layout considered, the row exchange allows the exchange of up to four products simultaneously instead of two as in the slot exchange, which has the potential to escape from local optima found by the slot exchange. Mathematically, consider a row located in (a_i, b_i) containing products k and l, i.e., $y^k_{a_i b_i} = y^l_{a_i b_i} = 1$, and another row located in (a_j, b_j) containing products m and n, i.e., $y^m_{a_j b_j} = y^n_{a_j b_j} = 1$, the row exchange reassigns k and l to (a_j, b_j), i.e., $y^k_{a_j b_j} = y^l_{a_j b_j} = 1$, and m and n to (a_i, b_i), i.e., $y^m_{a_i b_i} = y^n_{a_i b_i} = 1$. The row exchange is applied for all pairs of rows in different locations with products assigned to them. This neighborhood size is $O(|\mathcal{L}|^2)$ in the worst case, i.e., when there are products located in all existing rows.

The third neighborhood is defined as the swap of all products contained in different aisles. The *aisle exchange* LS finds two aisles with at least one ordered product in at least one of them and swaps their products. The position of the products in the original aisle is respected in their new aisle. This LS requires that the number of slots in the two swapped aisles are the same. Mathematically, for two swapped aisles $a_i, a_j \in \mathcal{A}$, given a product k located in (a_i, b), i.e., $y^k_{a_i b} = 1$ and another product l located in (a_j, b), i.e., $y^l_{a_j b} = 1$, the aisle exchange reassigns k to (a_j, b), i.e., $y^k_{a_j b} = 1$, and l to (a_i, b), i.e., $y^l_{a_i b} = 1$. This is done for all rows $b \in \mathbb{B}$ that belong to the two swapped aisles. For a warehouse with A identical aisles, this neighborhood size is $O(A^2)$ in the worst case.

2.5 Computational experiments

The SLAP and OPP literature lacks a well-established set of benchmark instances so that different methods could be compared on the same set. Instead, researchers usually create their own test instances based on the problem and the circumstances explored. We designed our instances using the parameters presented in Table 2.3 and share them on `https://www.`

leandro-coelho.com/slot-assignment-and-order-picking/. All instances consider the distances as $M = N = 1$. The small set is used to evaluate the MIP models and to adjust the GVNS parameters, while the regular set, consisting of instances similar in size to those used in the literature (Pansart et al., 2018; Scholz and Wäscher, 2017; Scholz et al., 2016), is solved only by the GVNS. In Table 2.3, column *Aisles* represents the number of identical two-sided picking aisles in the storage area, *Rows per aisle* is the number of rows in each aisle, each one containing two slots, *Orders* corresponds to the number of orders to be fulfilled, and *Products per order* is the number of unique products contained in each order to be picked in one pick round. The orders were generated using three different distributions, as shown in the *Demand* column. The *Random* distribution selects the products using a uniform distribution, while the *Skewed* distribution uses the Pareto principle to divide products into three classes. The *Skewed A/B/C* notation is used to state that items from the smaller group appear $A\%$ of the times, the medium group appears $B\%$, and the bigger group appears $C\%$. We used $A/B/C$ as either 50/30/20 or 80/15/5. Due to the low number of items in small instances, only the random distribution is used. Three different instances are created for each combination of these five parameters. The result is 108 small and 486 regular size instances. We highlight that the largest instances in the small set have the same parameters as the smallest instances in the regular set. However, they are generated with different seeds.

Table 2.3 – Warehouse layout and demand parameters used to generate the experimental instances

Set	*Aisles (A)*	*Rows per aisle (B)*	*Orders* ($\|\mathcal{O}\|$)	*Products per order* ($\|\mathcal{Q}_o\|$)	*Demand*
Small	1	5	1	3	Random
	3		5		
	5	10	10	5	
Regular	5	10	10	5	Random
	10		30	20	Skewed 50/30/20
	20	50	50	50	Skewed 80/15/5

Computational experiments are performed in an Intel Gold 6148 Skylake CPU with 2.4 GHz and with a memory limit of 16 GB of RAM. The linear MIPs are solved using the CPLEX C++ API (version 12.5). The routes for the SLOPP are generated using the Concorde TSP solver. As reported by Theys et al. (2010), the LKH heuristic approximates very well the optimal OPP solution. Concorde provides an implementation of the LKH heuristic, which is used here. We use previously computed routes for an order as a warm start to LKH. Based on preliminary experiments, we fixed the number of 4-swap kicks in the LKH heuristic to two kicks. The heuristic routing policies are implemented in C++.

2.5.1 Solving the linear MIP model

The first set of experiments consists in solving instances in the small set using the linear MIP models for the SLOPP and its four special cases. The problems are solved using CPLEX with

standard parameters and a time limit of two hours. The objective is to evaluate the maximum instance size for which an optimal solution can be proven using these models. For that, we group the instances from the small set according to the number of slots in the warehouse ($2AB$), and the number of items picked in a route ($|\mathcal{O}||\mathcal{Q}_o|$). Thus, each combination of Slots $\times$ Picks contains three instances. Table 2.4 shows the average time to solve the problem to optimality. Each cell is colored with a shade of grey representing the number of instances solved in that group, with darker cells representing more instances solved.

Table 2.4 – Average time to prove optimality of small instances using the linear MIP models for the SLOPP and its fours special cases

Problem	*Slots ($A \times B$)*	*Picks ($\|\mathcal{O}\| \times \|\mathcal{Q}_o\|$)*					
		3 (1 × 3)	*5 (1 × 5)*	*15 (5 × 3)*	*25 (5 × 5)*	*30 (10 × 3)*	*50 (10 × 5)*
SLOPP	*10 (1 × 5)*	0.3s	5.1s	-	-	-	-
	20 (1 × 10)	3.3s	-	-	-	-	-
	30 (3 × 5)	146.3s	-	-	-	-	-
	50 (5 × 5)	3435.6s	-	-	-	-	-
	60 (3 × 10)	7099.9s	-	-	-	-	-
	100 (5 × 10)	-	-	-	-	-	-
SL+Re	*10 (1 × 5)*	<0.1s	<0.1s	<0.1s	0.1s	3 0.1s	0.1s
	20 (1 × 10)	<0.1s	<0.1s	0.1s	0.9s	1.7s	22.8s
	30 (3 × 5)	<0.1s	<0.1s	0.8s	20.5s	83.1s	-
	50 (5 × 5)	<0.1s	<0.1s	4.6s	173.0s	181.9s	-
	60 (3 × 10)	<0.1s	<0.1s	0.7s	36.2s	216.5s	-
	100 (5 × 10)	<0.1s	0.1s	42.5s	294.3s	-	-
SL+Ss	*10 (1 × 5)*	<0.1s	<0.1s	<0.1s	0.1s	<0.1s	0.1s
	20 (1 × 10)	<0.1s	<0.1s	0.1s	0.7s	1.5s	21.7s
	30 (3 × 5)	<0.1s	<0.1s	1.0s	5.3s	296.3s	420.7s
	50 (5 × 5)	<0.1s	<0.1s	3.1s	21.6s	940.9s	-
	60 (3 × 10)	<0.1s	<0.1s	0.9s	15.1s	306.8s	-
	100 (5 × 10)	<0.1s	<0.1s	15.7s	57.0s	409.1s	-
SL+Mp	*10 (1 × 5)*	<0.1s	<0.1s	0.1s	0.2s	0.9s	1.4s
	20 (1 × 10)	<0.1s	<0.1s	1.2s	4.2s	10.7s	210.8s
	30 (3 × 5)	0.1s	0.1s	136.4s	1199.1s	-	-
	50 (5 × 5)	0.1s	0.2s	862.6s	4321.4s	-	-
	60 (3 × 10)	0.1s	0.2s	447.0s	3174.8s	-	-
	100 (5 × 10)	0.3s	0.9s	2984.6s	-	-	-
SL+Lg	*10 (1 × 5)*	<0.1s	<0.1s	0.4s	0.8s	2.5s	4.8s
	20 (1 × 10)	0.1s	0.1s	2.0s	25.4s	54.6s	6453.6s
	30 (3 × 5)	0.1s	0.2s	28.1s	71.6s	896.6s	-
	50 (5 × 5)	0.2s	0.3s	148.2s	387.2s	-	-
	60 (3 × 10)	0.4s	0.9s	155.1s	3018.8s	-	-
	100 (5 × 10)	1.5s	2.0s	656.0s	5044.5s	-	-

*Each cell contains the average over three instances. Dark gray shows that all three instances are solved. Medium gray indicates that only two instances are solved to optimality, and light gray indicates that only one instance is solved to optimality

The results show that the linear MIP model for the general SLOPP is of no practical use even for small instances with only three or five picks, which have optimal solutions that could be found intuitively. We also highlight how fast the average time to prove optimality increases by adding more slots to the warehouse, even with only three picks. In larger instances, the SLOPP model runs out of memory, which is in line with Forrester (2016) about the standard linearization technique requiring a huge number of variables and constraints. The linearization

of the cubic term in the objective function in the SLOPP model requires the introduction of $O(|\mathcal{O}||\mathcal{L}|^2|\mathcal{P}^2|)$ new variables and constraints, which for large instances of the small set represents hundreds of millions of new variables.

When solving the models for the special cases, the solver was able to find optimality in most of the small instances. The SL+Re and SL+Ss models, which are relatively simpler in terms of number of variables and constraints, can be solved for instances slightly larger than the SL+Mp and SL+Lg models. We note that although there are cubic terms in some of the objective functions, they require the addition of only $O(|\mathcal{O}|A)$ new variables and constraints, significantly fewer than in the general SLOPP.

2.5.2 GVNS setting

Since the MIP models are difficult to solve for larger instances from the small set, an alternative method to deal with instances from the regular set is required. We now show how the GVNS was set for the experiments to solve the regular size set.

Setting the search strategy

In the neighborhood search strategy setting, we demonstrate through experiments the reason why the application of the three neighborhoods proposed in our VND is more effective than the application of only one or two of them. The experiment consists in creating initial solutions for the storage location subproblem of the SLOPP using the three storage policies implemented and by applying VND for each of them using different combinations of LSs. We tested an improvement strategy of using a single, a pair and a trio of LSs in all possible orders exhaustively, i.e., until no improving solution can be found. Results are compared for all instances of the small instance set for which the MIPs proved optimality. A table summarizing the results obtained is presented in A.1. The results indicate that the average gap significantly decreases when more neighborhoods are explored, when all three of them are performed, and the exploration order is not important (unlike when only two of them are used). For this reason, and to increase diversity, for the remaining experiments, we set the neighborhood exploration order to be randomly decided at the beginning of every VND. This leads to a reduction in the search bias during GVNS cycles, which helps avoid local minima. We also note that the exploration of the three neighborhoods is performed within one second, even for the largest instances of the set.

Setting the number of shakes and cycles

We solve the SLOPP and its special cases for all small instances using the proposed GVNS to improve solutions generated by the first VND. From the experiments described and with results reported in A.2, we set the maximum number of shakes $S = 5$. Then, each run is performed again up to $K = 20,000$ cycles. We calculate gaps to the BKS, either a proven

optimal or the best feasible solution known, at different parts of the GVNS. The first solution observed is the initial solution. The next solution observed is that after VND is applied to improve the initial solution (Algorithm 3, line 2). Then, we observe the current best solution for several values of K. From 10 different runs using different initial solutions generated using the storage policies implemented, a very fast convergence is observed when GVNS is at $K = 20$, so that the difference between solutions for all runs is no more than 0.01%. Since all solutions converge to similar ones regardless of the initial one, this is evidence of the robustness of this metaheuristic when improving solutions for the SLOPP and its special cases. The results of these tests indicate that solutions are improved up to $K = 10,000$, within a run time of less than two minutes. For these reasons, we choose this setting for the remaining experiments.

2.5.3 Solving the regular size set

We now perform detailed experiments to evaluate our algorithm on the regular size instance set. We analyze the solutions obtained for the SLOPP when different routing policies are used to create the picking routes. First, we use a method where solutions are evaluated during the whole search by a single routing policy. Four heuristic policies – return (Re), S-shape (Ss), midpoint (Mp), and largest gap (Lg) – and an optimal policy (LKH) are considered resulting in solutions for each of the five problems – SL+Re, SL+Ss, SL+Mp, SL+Lg and SLOPP, respectively – considered here. After, we present a new strategy that alternates the routing policy between Mp and LKH to show how the GVNS can benefit from the speed of the latter and the quality of the former to generate even better solutions for the SLOPP. The GVNS is run with the previous determined parameters ($S = 5$ and $K = 10,000$). The initial solution used is the best among the three generated from the storage policies. Each instance is solved five times. A two-hour time limit is used in all experiments since managers usually dispose of enough time to make rearrangement decisions when they are done, for example, on a daily basis. The results are presented for each problem setting, so that it is possible to analyze their impact in combination with each routing policy.

Comparison of solutions for the SLOPP and its special cases using GVNS with different routing policies

In this round of experiments, the objective is to compare the GVNS performance using different routing policies given the same run conditions, i.e., the same run time limit and parameters. We remind that a solution given by GVNS with any heuristic routing policy (Re, Ss, Mp, Lg) is both a solution for the special case and the integrated SLOPP, since the latter generalizes the former. From the results, we can draw conclusions about which special case gives better solutions under different instance conditions, and also for what routing policy the GVNS finds better solutions for the SLOPP. Table 2.5 shows the average route length given by GVNS with routes created using the five routing policies. The table is complemented with A.3, where we show the average run time and last cycle performed before the search stops. We have grouped

all instances with different demand distributions in each row of the table, presenting the results for combinations of the remaining instance attributes. So, it is possible to identify for which routing policy the GVNS finds better solutions under each combination of warehouse size and orders' characteristics.

We conclude from the experiments that the best policy to generate routes for the SLOPP in our GVNS in most cases is Mp since these solutions consistently outperform the others. They have the lowest average route length and the largest number of best average solution for combinations of instance attributes. The second best is the Re, which presents a good performance, specially in warehouses with long aisles ($B = 50$). LKH finds some of the best solutions in smaller warehouses ($A = 5$ and $B = 10$), but significantly worse solutions in larger ones. Finally, Ss and Lg did not have the best average solution in any case.

The key factors to explain the success of midpoint compared to the other policies are: (i) low computational complexity; (ii) differentiated special cases; and (iii) good approximation of optimal routes.

Regarding factor (i), as shown by A.3, Mp average running time is slightly higher than those of Re and Ss, but lower than that of Lg and significantly lower than that of LKH. Lower computational complexity means that more solutions can be evaluated within the time limit, as shown by the higher value of k in A.3. Consequently, more neighborhoods are explored during the search, which increases the chances of finding the global optimum solution. The high computational complexity is the possible explanation of LKH performing well in many small instances ($A = 5$ and $B = 10$) and deteriorating fast when the instance becomes larger.

Factor (ii) differentiates the performance of Mp when compared to Lg. Normally, it is expected that Lg should dominate Mp (Dijkstra and Roodbergen, 2017), since it is a less restricted version of that. Since good SLOPP solutions tend to concentrate high demanded products close to the I/O point, it is common that solutions contain orders with all picks in two or more aisles of rows located across the front cross aisle. Examples of this are commonly found in instances with few picks ($\mathcal{Q}_o = 5$) and long aisles ($B = 50$). In this case, Re generates a better route than Lg or Ss. However, when $\mathcal{Q}_o$ is higher, some orders may require that products are picked further into the aisles. In this case, it may be better to entirely cross both aisles, as one would under Ss and Lg. The more aisles a storage area has ($A = 20$), and the more routes are performed ($\mathcal{O} = 50$), the greater are the chances that entirely crossing two aisles is better. Since Mp can provide good solutions to both situations, it has a higher chance of finding better solutions overall. Even if its special cases were incorporated to Lg, there is no guarantee that it would outperform Mp due to factor (i).

Factor (iii) differentiates Mp from Re and Ss. These two policies are known to perform well only under very special circumstances (Dijkstra and Roodbergen, 2017; Petersen, 1997; Van Gils et al., 2018b). The return policy approximates well an optimal route when picks

Table 2.5 – Average route length found by GVNS with different routing policies

A	B	$\mathcal{O}$	$\mathcal{Q}_o$	*Re*	*Ss*	*Mp*	*Lg*	*LKH*
5	10	10	5	**108.9**	144.7	**108.9**	144.7	109.2
			20	350.5	317.7	292.3	302.8	**290.5**
			50	717.3	542.8	549	558.4	**522.0**
		30	5	502.5	612.4	**496.8**	608.6	501.9
			20	1466.8	1250.5	1142.0	1174.9	**1115.7**
			50	2534.1	1872.4	1979.0	1940.9	**1739.5**
		50	5	999.5	1159.1	**966.7**	1146.0	968.5
			20	2694.8	2256.7	2080.0	2121.0	**2018.8**
			50	4412.8	3228.8	3468.0	3359.8	**2994.6**
	50	10	5	113.4	134.3	**113.2**	133.2	115.5
			20	374.2	808.4	**372.9**	822.6	708.1
			50	**922.5**	1143.1	947.9	1083.0	1089.1
		30	5	583.8	972.8	**581.6**	994.0	602.8
			20	**2222.4**	3102.0	2386.4	3134.1	2931.6
			50	4922.0	5224.7	**4030.5**	4347.0	4587.1
		50	5	1303.2	2264.4	**1297.7**	2264.8	1352.0
			20	4992.7	5986.1	**4899.6**	5613.6	5324.8
			50	10400.8	9629.5	**7852.2**	8379.5	8578.6
10	10	10	5	**110.0**	134.9	**110.0**	134.9	110.9
			20	365.6	358.5	**320.4**	323.9	323.8
			50	797.8	709.6	**668.0**	687.6	669.0
		30	5	514.9	606.2	**511.7**	604.2	522.3
			20	1685.7	1619.3	1441.3	1466.9	**1414.4**
			50	3265.1	2701.9	2632.9	2627.2	**2526.6**
		50	5	1058.6	1235.4	**1048.8**	1215.8	1063.8
			20	3298.3	3100.2	2718.4	2758.7	**2674.4**
			50	6044.4	4883.4	4833.7	4764.4	**4562.5**
	50	10	5	114.9	119.3	**114.8**	119.3	116.5
			20	325.2	826.3	**325.0**	824.8	925.4
			50	**788.4**	1167.6	807.7	1068.1	1566.4
		30	5	**542.4**	733.5	543.1	743.2	560.1
			20	1898.9	3261.7	**1876.1**	3251.6	3175.4
			50	4775.0	6662.5	**4559.0**	4666.0	5809.5
		50	5	1154.8	1726.9	**1153.2**	1757.6	1191.1
			20	**4330.3**	6549.2	4871.5	6029.2	5780.8
			50	10877.9	12987.1	**8957.8**	9730.3	11925.9
20	10	10	5	112.8	125.1	**112.7**	125.1	114.3
			20	360.8	378.7	**336.5**	337.3	349.7
			50	854.7	815.0	**726.1**	770.5	795.6
		30	5	525.2	585.0	**521.8**	582.0	535.8
			20	1813.4	1918.1	**1623.8**	1682.3	1657.9
			50	3845.7	3612.3	**3308.8**	3337.2	3317.5
		50	5	1100.0	1237.3	**1086.0**	1218.4	1118.7
			20	3697.3	3861.1	**3226.8**	3307.1	3331.5
			50	7572.4	6902.4	6323.3	6330.7	**6256.9**
	50	10	5	116.7	116.9	**116.7**	116.9	118.4
			20	**321.7**	723.5	321.9	757.3	1006.7
			50	788.2	1227.8	**786.9**	1077.9	1873.5
		30	5	**557.2**	580.3	558.1	588.0	565.3
			20	1638.4	3262.6	**1629.2**	3257.4	2807.9
			50	**4209.3**	7547.3	4526.6	4991.3	6864.4
		50	5	1163.6	1347.8	**1163.2**	1361.4	1184.5
			20	3710.3	6742.5	**3696.5**	6051.7	5775.2
			50	9874.4	15597.6	**9500.8**	10422.8	13777.4
Average				2293.2	2715.1	**2048.6**	2355.9	2443.0

are located either in only one aisle or in many aisles if they are in locations across the front cross aisle. Thus, Re is particularly good in warehouses with long aisles ($B = 50$) as shown by the experiments. Meanwhile, the S-shape policy performs best when there are many picks distributed along the entire storage area. Since optimal storage location assignments tend to concentrate high demanded products close to the I/O point, this situation arises when all stored products are highly demanded, such that every single order would contain picks in many different locations around the storage area. In our instances, this is the case of small warehouses ($A = 5$ and $B = 10$) with many products per order ($\mathcal{Q}_o = 50$).

Other observations from the results are:

- instances with few picks and large warehouses (e.g., $A = 20$, $B = 50$, $\mathcal{O} = 10$, $\mathcal{Q}_o = 5$) have similar solutions regardless of the routing policy, since the optimal routes in those cases are the "go to the aisle, pick and return" special case. In these scenarios, picks from a single order are usually concentrated close to each other in a single aisle;
- in horizontal layouts, i.e., with a large number of short aisles ($A = 20$ and $B = 10$), and for a large number of picks ($\mathcal{Q}_o = \{20, 50\}$), Lg and Mp perform best. In vertical layouts, i.e., with a small number of long aisles ($A = 5$ and $B = 50$), Mp and Re are the best. Possible explanations are such as those previously discussed for factor (ii);
- in warehouses with a similar layout proportion ($A = 5$ and $B = 10$ against $A = 20$ and $B = 50$) for the same number of picks, solutions are usually better in the smaller warehouse. The reason is that in our instances smaller warehouses have fewer varieties for ordered products and, consequently, a higher average demand per product, meaning a higher concentration of picks near the I/O point.

Solving the SLOPP using GVNS with a combination of Mp and LKH

From the previous experiments, we observe the robustness of Mp since it performs well in our GVNS under different instance settings. The results show that instead of wasting time evaluating routes for bad SLAP solutions, it is better to use a fast and fairly good heuristic to find better storage location assignments quicker, speeding up the improvement of the initial solution. However, it is expected that Mp can provide a good approximation to optimal OPP solutions, but not as good as LKH, given the same SLAP solution. Thus, we suggest here a different search strategy for our GVNS that alternates between Mp and LKH to solve the OPP sub-problem. This strategy consists of using Mp during the most intensive part of GVNS, i.e., the VND, and then update the solution using LKH to optimize the routes when VND is over. This way, the routes for the local optimum SLAP solution generated in the VND using Mp are updated to near optimal ones with LKH. We performed a new round of experiments for this new strategy. Detailed results are reported in A.4. In summary, it shows that the

average solution provided by GVNS is improved from 2048.6 to 2023.5 (1.2% of improvement) when compared to the average solution of using only Mp. Since LKH requires more time to be solved, the average number of cycles to run this combined Mp/LKH is lower than when using only Mp. For that reason, there are situations where Mp alone may still outperform the combined Mp/LKH.

Solution improvement over the change of the storage policy

We analyze the potential improvement on route length by changing the storage policy from a heuristic to the optimal one considering that the routing policy is maintained. For each instance of the SLOPP, solved using the Mp/LKH strategy, and its four special cases, solved using the respective heuristic routing policy, we consider two solutions obtained in the experiments from Section 2.5.3 and 2.5.3: (i) the best initial solution among the three generated using a storage policy (random, diagonal and within-aisle); and (ii) the average solution of the five runs of GVNS for the storage location with the respective routing policy. In Table 2.6, we present the improvement from (i) to (ii) in column *Improv. over initial* calculated as $improv(x) = \frac{f(x) - f(x')}{f(x)}$, where x is the solution (i) and x' is the solution (ii). In each line of the table, average results are presented for the parameters shown on the left.

Table 2.6 – Average solution improvement from the heuristics to an optimal storage policy for the SLOPP and its special cases

		SL+Re	*SL+Ss*	*SL+Mp*	*SL+Lg*	*SLOPP*
		Improv. over initial (%)	*Improv. over initial (%)*	*Improv. over initial (%)*	*Improv. over initial (%)*	*Improv. over initial (%)*
A	*5*	40.7	28.8	42.1	33.7	35.1
	10	51.0	40.5	52.6	46.0	45.9
	20	**59.0**	**50.2**	**58.7**	**53.8**	**53.8**
B	*10*	38.1	32.9	41.0	35.8	33.1
	50	**62.4**	**46.8**	**61.3**	**53.2**	**56.8**
$\mathcal{O}$	*10*	**56.5**	40.9	**55.9**	45.3	**50.5**
	30	49.6	**41.0**	50.7	**45.9**	44.2
	50	44.6	37.6	46.8	42.3	40.0
$\mathcal{Q}_o$	*5*	**53.6**	**55.0**	**53.3**	**53.4**	**52.1**
	20	53.2	36.7	52.9	38.5	47.3
	50	44.0	27.9	47.1	41.5	35.4
D	*Random*	**55.4**	**45.9**	**55.4**	**50.1**	**49.7**
	Skew 50	51.9	41.1	52.8	46.6	46.4
	Skew 80	43.4	32.6	45.2	36.8	38.7

Two main conclusions are drawn from these results. First, regardless of the problem considered, the search for optimal storage location assignments significantly improves the solutions generated by the heuristic storage policies, with average improvements from 27.9% (SL+Ss and $\mathcal{Q}_o = 50$) to 62.4% (SL+Re and $B = 50$). Finally, the total distance traveled is improved more when optimal storage locations are searched, compared to using heuristic storage policies, when warehouses are larger, when there are fewer orders or orders with fewer items, and

when demands are less skewed. This pattern is observed in almost all routing scenarios tested as boldened in the table.

2.6 Practical notes and further discussions

The results from research and practice show the importance of storage location and picker routing planning on the order picking performance. Traditionally, these problems are solved separately since the storage location has been considered to be more static in the literature and assumed to be less prone to changes. However, several factors, such as seasonality, marketing efforts, shelf life, and expiration dates, have an impact on the demand of the products. This means that the SLAP solution, in certain industries and for certain businesses, would be more dynamic and should be revised periodically. Some may argue that the cost of relocations is too high to justify product location reassignments being performed more often. However, considering the travelled distance minimization, the results of our experiments show the benefits of changing storage and routing policies in order to improve picking efficiency. Therefore, although known intuitively, warehouse managers should consider this link between the two problems, as modeled in our SLOPP.

Although it is interesting to see that mathematical models can be used to solve different versions of the SLOPP, the experiments show that they have limitations to solve even small instances. As our results show, the time spent to solve the problem increases dramatically as the size of either the warehouse or the orders grow. In reality, warehouse managers may deal with thousands of products in very large warehouses. The size of these real instances makes those exact methods impractical. Therefore, an application of a solution algorithm such as our GVNS not only saves time – as the unsolvable instances by the exact method are now solved within less than a second – but also provides good quality solutions in practice.

Obviously, when one changes from heuristic policies to optimal algorithms, it is expected to achieve better solutions. In this paper, we considered a case with single-block layout, two-sided picking and known orders where using the optimal storage layout we could on average reduce between about 27% to 62% of the total route length traveled by the pickers. Larger warehouses with fewer routes and smaller orders are especially fit for this change. From the warehouse managers point of view, the tradeoffs between reshuffling costs and the savings obtained by changing the current storage assignments to meet the SLOPP solution obtained by our GVNS need to be evaluated. Warehouse reshuffling policies and costs are discussed in Pazour and Carlo (2015). Since reshuffling can be performed in several ways depending upon the storage and retrieval system in use, this is an interesting topic for further investigation. Future research can consider the integration of SLOPP with the reshuffling problem for specific systems.

When rearrangements are performed less frequently than on a daily basis, say weekly or

monthly, managers may dispose of much more time than the two-hour time limit considered in our experiments (Kofler et al., 2014). Since most of the runs were stopped before reaching the number of cycles so that solutions converge, this indicates that better solutions are still within reach if we could let our GVNS run for a longer time. In order to prove this claim, we ran experiments for the SLOPP using our GVNS with the Mp/LKH strategy for eight hours, and we observed an improvement of about 1% on average when compared to the two-hour limit (avg. solution reduced from 2023.5 to 2002.7), with improvements of up to 3% being found for larger instances. Nevertheless, the low improvement is an indication that convergence, and possibly the optimal solution for the problem, is not far. Finally, although not tested here, other combinations, possibly even more complex, can be developed for specific cases that may outperform both Mp and Mp/LKH for the SLOPP.

2.7 Conclusions

In this paper we have introduced new optimization tools to solve storage location assignment problems. The SLAP consists in arranging the products in a warehouse in order to optimize material handling during the picking process. A SLAP solution is evaluated according to the performance of the order picking, measured as the distance traveled by pickers. When decisions regarding product assignment and routing are integrated, we have the storage location and order picking problem (SLOPP) as introduced here. The SLOPP is modeled as a cubic mixed integer programming problem. We also model four special cases of the problem when routing policies are imposed, i.e., when the routes must be created using a heuristic routing policy, which are straightforward methods to create picking routes. These policies are modeled mathematically considering a typical single-block warehouse for the first time in this paper. To solve the SLOPP, we assume that all orders are known *a priori*, which might not be the case in markets with too volatile demands. Otherwise, when demands are known at the order level, the integration of these problems is advantageous.

Extensive computational experiments show that the models can deal with very limited size instances. As an alternative, we have proposed a General Variable Neighborhood Search (GVNS) metaheuristic with a descent phase consisting of the exploration of three different neighborhoods adapted for the problem. The GVNS presented is capable of providing optimal solutions for all instances solved using the models within less than a second. We also show that, in instances of size comparable to those of real scenarios, the search for optimal storage location assignments leads to savings ranging from about 27% to 62% compared to solutions from common storage policies. Savings can be even higher when considering that routing policies can also be changed, for example, from a heuristic policy to an optimal one. GVNS has also the potential to keep improving solutions the longer it runs. This demonstrates the potential benefits of searching for an optimal assignment of products to storage locations.

Research directions include strenghtening or remodeling the formulations to speed up the search for the optimal solution in the single-block layout and other ones. Finally, other elements from warehouse planning may be incorporated to the SLOPP, such as the integration with other decisions (batching, zoning), constraints (safety, blocking) and parameters (rearrangement costs to modify the storage plan).

Chapter 3

Estimating optimal ABC zone sizes in manual warehouses

Chapter information A paper based on this chapter is currently submitted for publication: A. Silva, K. J. Roodbergen, L. C. Coelho, and M. Darvish. Estimating optimal ABC zone sizes in manual warehouses. Document de travail du CIRRELT-2021-22, Tech. Rep., 2021.

Résumé

Quand les demandes futures ne peuvent pas être prévues avec précision, c'est commun l'utilisation d'une politique de stockage ABC pour le problème d'affectation des produits. Dans cette politique, la zone de stockage est divisée en trois zones, puis des ensembles de produits les plus demandés sont affectés aux zones avec les meilleurs emplacements de stockage. Malgré la popularité de cette méthode, les dimensions des zones sont encore choisies arbitrairement en pratique, menant à une grande perte d'efficacité dans des entrepôts. Dans ce chapitre, nous identifions les facteurs d'entrepôt parmi l'aménagement, les caractéristiques de la demande et les politiques pour le zonage et le routage, qui influencent les dimensions optimales des zones dans une politique ABC, c'est-à-dire, celles qui minimisent la distance moyenne attendue des routes. Nous proposons l'utilisation de modèles d'apprentissage automatique pour prédire les dimensions optimales des zones en considérant les plus pertinents parmi les facteurs mentionnés. Ces modèles sont entraînés à l'aide de données générées à partir de simulations des performances des entrepôts sous de nombreuses dimensions de zones. Des expériences montrent que les dimensions fournies par nos modèles améliorent considérablement l'efficacité du prélèvement des marchandises par rapport aux dimensions arbitraires couramment utilisées, notamment pour les systèmes à une seule zone (politique aléatoire), à deux zones (règle 20/80) et à trois zones (20/30/50).

Summary

In situations where future demands cannot be precisely forecast, it is common to use the class-based ABC storage policy for the storage location assignment problem. In this policy, the storage area is divided into three zones, then sets of most-demanded products are assigned to the zones with the best storage locations. Despite the method's popularity, zone sizes are still mostly chosen arbitrarily, which can lead to major efficiency loss in many common warehouse settings. In this chapter, we identify warehouse factors among the layout, demand characteristics, and policies to define zones and picking routes, that influences the optimal zone sizes in an ABC policy, i.e., minimizes the expected average route length. We propose the use of machine learning models to predict optimal zone sizes considering the most relevant among the mentioned factors. These models are trained using data generated from simulations of the performance of common warehouse settings under many zone sizes. Computational experiments show that zone sizes provided by all models significantly improve the order-picking efficiency when compared to the arbitrary zone sizes commonly used, notably for the one-zone (random policy), the two-zone (20/80 rule), and the three-zone (20/30/50) systems.

3.1 Introduction

When products from a supplier arrive at a warehouse, they often stay temporarily in a storage area, such as the forward or reserve areas. Each of these areas may still be divided into different areas to accommodate products, e.g., in pallet racks or on shelves, according to their characteristics, such as size, temperature, etc. Products remain in these storage areas until they are retrieved, in response to customers' orders. Methods for *storage location assignment* decide the location within a storage area to assign each product (Reyes et al., 2019). Typically, a warehouse receives products from a supplier in bulk (for example, a full pallet), while shipments to customers are in significantly smaller quantities (for example, in boxes). Hence, the retrieval of products from storage, called *order-picking*, tends to be much more labor-intensive than their storage. Generally, order-picking is the most time-consuming activity of the entire warehouse (Frazelle, 2016). Furthermore, the order-picking process is often time critical, for example, due to strict deadlines for meeting same-day or next-day delivery requirements in the e-commerce (Boysen et al., 2019a). It is for these reasons that the design of good policies for storage location assignment is vital for achieving an efficient order-picking process in warehouses (De Koster et al., 2007; Petersen et al., 2004).

If the storage and order-picking processes are performed by workers who walk or drive between locations in a storage area, this is commonly referred to as a *manual warehouse* or a picker-to-parts warehouse (De Koster et al., 2007). We describe the warehouse as being "manual" to refer to the system used in the storage area of interest in this study. Of course, the same warehouse may contain multiple storage areas being operated by different systems,

either manual or automated. Despite the significant attention given to automated warehouse systems in the literature (Boysen et al., 2019a), manual warehouses can provide better flexibility for accommodating demand fluctuations, and have lower investment costs (Calzavara et al., 2019). Flexibility and scalability are especially valuable in warehouses of online retailers (Schrotenboer et al., 2017). These emphasize the need for continued process improvements for manual warehouses. The storage area of a typical manual warehouse has a rectangular shape consisting of parallel pick aisles. The area is divided into blocks and pickers use cross aisles between pairs of blocks to change aisles. Pick routes start and end at an I/O point located at one corner of the storage area. The layout configuration involves a number of strategic decisions such as the number and size of aisles and blocks. A graphical display of such layout is provided in Silva et al. (2020, Figure 3).

Storage location assignment problems are among the tactical level decisions. Policies for this problem match products with locations based on their properties. The 'most important' products should be assigned to the 'best' locations, and the 'least important' products to the 'worst' locations. For deciding which products are 'most important', the criterion of order frequency is commonly used, i.e., how often a product appears on a customer's order (e.g., Guo et al., 2016; Lee and Elsayed, 2005; Petersen, 1999). However, order frequency is not stable over time, which may induce the need for repositioning products when their demand has changed (Pazour and Carlo, 2015). To mitigate the need for repositioning, while simultaneously maintaining the advantages of assigning 'most important' products to 'best' locations, the concept of *class-based storage* is widely deployed in practice and studied in the literature (e.g., Chan and Chan, 2011; Muppani and Adil, 2008). In the class-based storage, products are grouped into classes, based on order frequencies, and each class is subsequently assigned to a dedicated *zone* of the warehouse. Within a zone, the assignment is random. The class of the fastest moving products is generally called class A, the next fastest-moving class is called class B, and so on. Since each product is assigned to only one zone, scattered storage is not commonly used in class-based policies. We still highlight that the meaning of zones in the storage location assignment context, as used here, differs from zones in the order picking context. In the latter, the storage area is divided in pick zones where a set of pickers are dedicated to a single zone, and orders are split among different pickers (Van Gils et al., 2018c), where workload balance becomes a relevant factor (Vanheusden et al., 2020). Although zone picking is not considered here, this study can be easily extended for it.

To implement a class-based storage system, three decisions must be made: (i) the number of classes, (ii) the size of each zone for each class, and (iii) the positioning of each zone in the layout. Their objective is to reduce the expected average order-picking time, which is a function of the distances traveled by pickers to retrieve products, referred here as the average route length (ARL). From the literature, it is known that adding more classes tends to decrease the ARL, albeit at a decreasing rate (De Koster et al., 2012). Since having a

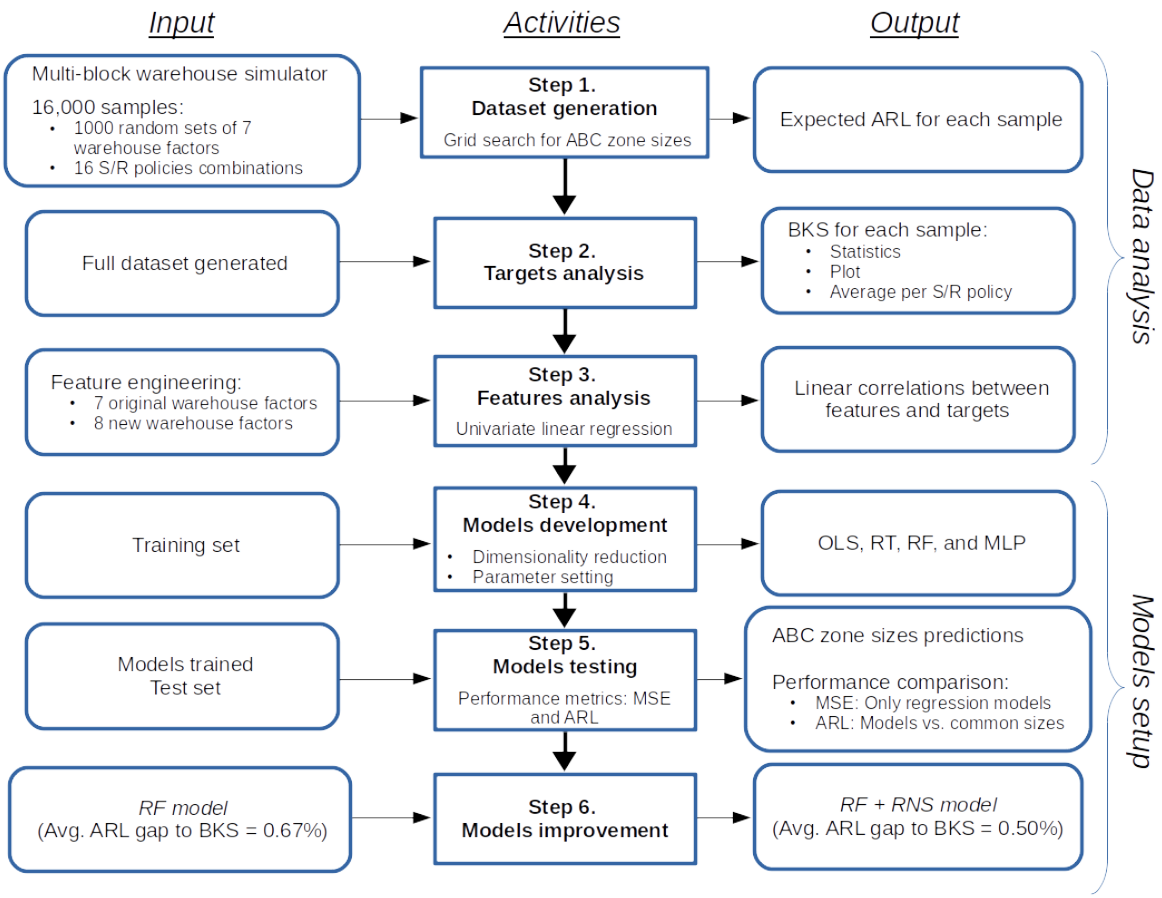
large number of classes defeats the purpose of hedging against demand frequency changes, a common compromise between the two objectives is to use three classes (see, e.g., Hausman et al., 1976; Roodbergen, 2012; Mirzaei et al., 2021), which is often referred to as the *ABC storage*. Also, the decision problem of where to position each zone is extensively studied (see, e.g., Chan and Chan, 2011; Le-Duc and De Koster, 2005; Petersen et al., 2004). In contrast, zone sizing has received scarce attention in the literature. In Section 3.2, we review relevant literature in more detail. Note that methods to address the positioning of zones are often referred to as *storage policies* (Petersen, 1999; Silva et al., 2020). Even though this wording may appear more encompassing than it actually is, we adhere to this convention.

Zone sizes depend on a multitude of strategic factors, including the warehouse layout, and tactical factors, such as operating policies that impact the picking process. Notably, relevant operating policies to consider are storage policies (how the zones are positioned in the layout) and routing policies (how a route through the warehouse is determined to retrieve the products in a pick list). Computing the actual ARL in each route is not possible when the decision on zone sizing is made since the routes that pickers will follow depend on the actual pick lists, which are unknown at the tactical level. A challenge is that there are no closed-form mathematical expressions to determine the expected ARL for every possible warehouse setting. The only alternative for a consistent performance analysis is through simulation. A straightforward method to solve the zone sizing problem is by simulating a warehouse with many combinations of zone sizes and observing which one leads to the minimum ARL. This has been done in the literature in small scale only (e.g., Petersen et al., 2004). However, simulation requires relatively high computation times, which makes it intractable to compare very large number of solutions. For this reason, the adoption of arbitrary zone sizes is a common practice.

The first major contribution of this paper is to implement a tailor-made simulator, extending the work of Roodbergen (2012). The simulator considers many factors derived from the layout properties of the multi-block warehouse, the demand frequency of products, the number of products in customers' orders, and the storage and routing (S/R) policies in use. Since the number of possible settings is infinite, we generate a limited number by randomly sampling from an extended range of commonly occurring settings, including shelf rack layouts (Çelik and Süral, 2019) and pallet rack layouts (Parikh and Meller, 2010). Each setting is simulated considering thousands of zone sizes using grid search, and each zone size is evaluated by their expected ARL. Even for a limited number of settings, the extensive simulations required about 12,000 CPU hours. We analyze the big data generated to quantify the influence of many new input factors (features) in the best zone sizes (targets) using a univariate linear regression. We also show that the best observed zone sizes significantly change according to the warehouse settings, therefore justifying the need of deploying tools that can estimate the optimal sizes given a warehouse setting.

As a second major contribution, we propose a new methodology to solve the zone sizing problem using machine learning (ML) regression methods. The ML methods result in models that receive as input the features previously mentioned and output the three zone sizes required for the ABC storage. Although in preliminary experiments nine ML regression methods were tested, only the four most promising ones are presented. They are ordinary least squares (OLS), regression tree (RT), random forest (RF), and multilayer perceptron (MLP). Their models are developed using the data generated in the simulations. Briefly, we select the zone sizes that lead to the lowest ARL for each warehouse setting simulated, and these zone sizes are used as target values to train the regression models. We remove unnecessary features through a dimensionality reduction step and set the relevant parameters of the models. The models are compared against themselves and the most common zone sizes used in practice, including the use of a random policy, a two-zone (20/80) policy, and a classical 20/30/50 policy, by their performance using the mean square error (MSE) metric and the ARL. We also show that an additional step of rounding predictions to the nearest subaisle (RNS) can improve the solutions. Finally, we show that there is a trade-off between simplicity and performance among the ML models trained. Figure 3.1 presents a summary of the methodology described.

Figure 3.1 – Summary of the methodology adopted in this paper



By deploying ML methods on this dataset, we aim at developing models that can quickly and easily determine proper zone sizes, given any set of input factors, without future need for repeating the extensive simulations. We highlight that our methods do not imply that the zone sizing decision should be done after other strategic and tactical decisions. Also, this methodology can be easily adapted for different problems found in warehouses, both manual and automated.

The remainder of this paper is structured as follows. Section 3.2 gives an overview of related literature. In Section 3.3, we describe our simulation model, including a detailed description of the factors considered. Section 3.4 contains an overview of the four ML methods used. Section 3.5 presents an analysis of the data gathered from the simulations. Section 3.6 presents the setup of the regression models, their performance comparison, and a qualitative analysis of each method attribute. Finally, the concluding remarks of this paper are found in Section 3.7.

3.2 Literature review

Zone shapes and sizes are usually investigated together in the class-based storage literature. For warehouses with single-command cycles, in which storage and retrieval operations are performed carrying a single load, found mainly in automated systems and unit-load warehouses, these problems can be solved analytically. Hausman et al. (1976) analyze the optimal partitioning point for a two-zone system in terms of the demand skewness, showing that the first zone should be smaller when demands are more skewed. For the ABC system, the best partitioning points are determined numerically using grid search. The results show that the best combination of classes A and B tends to be smaller when demands are more skewed and, consequently, class C tends to be larger. Eynan and Rosenblatt (1994) extend the Hausman et al. (1976) procedure to optimally determine zone boundaries for n classes. Rao and Adil (2017) also develop analytical models to find an optimal partitioning point in a two-zone system, but considering that storage location assignments are determined using a heuristic storage policy. Similar procedures are also presented for different systems, such as for a 3D compact automated system (Yu and De Koster, 2009), a live-cube compact system (Zaerpour et al., 2017), and a warehouse with diagonal cross aisles (Bortolini et al., 2019b). Still for single picks, Van den Berg (1996) solves the zone sizing problem using a dynamic programming algorithm that simultaneously assigns locations and products to classes.

In this paper, we consider a setting where multiple picks are performed in each route. Different than for single-command cycles, there is no firm strategy on how to define class partitions in a multiple picking setting (De Koster et al., 2007). For a specific case of a warehouse with a single-block layout, Petersen et al. (2004) analyze how to set up the class-based policy for zones shaped following simple rules commonly used in practice in a system with two, three and four zones. A few different zone sizes are tested only for the two-zone case, which they

concluded that either a 30/70 or 40/60 partition is the best option for the specific layout considered, depending on the number of picks to be done in the routes. For the three-zone case, they arbitrarily considered sizes as 20/30/50. In fact, for many studies, the size of each zone is an input parameter in their models, not a decision variable (Dijkstra and Roodbergen, 2017; Manzini et al., 2015; Roodbergen et al., 2015; Sooksaksun et al., 2012). Many of them consider that demands follow a 20/80 curve (20% of products account for 80% of the total demand) and, therefore, define zones using the 20/80 partition for two classes or the 20/30/50 for three classes. Using these zone sizes for a skewed demand scenario in an ABC system, Le-Duc and De Koster (2005) investigate zone shapes in a two-block warehouse with the I/O located in the head of the cross aisle that separates them. A mathematical model is developed to determine the partial length of each aisle used for storing each class. They show that the shape of the optimal zones depends largely on the demand skewness, the number of picks in each route, the storage assignment policy, and the warehouse length/width ratio. For the same two-block layout, Rao and Adil (2013) develop analytical models to simultaneously determine the number of classes, class boundaries, pick list size, and number of aisles. Dijkstra and Roodbergen (2017) show that the optimal class boundaries in a single-block warehouse where pickers follow the return routing policy must be non-increasing as a function of the aisle number. Chan and Chan (2011) present a case study of a single-block, multi-level rack warehouse with an ABC storage system where different combinations of S/R policies are simulated for various pick densities. Roodbergen (2012) provides insights on the interactions between layout, routing, and storage policies in an ABC system, showing that the best zone sizes change according to the policies in use. We extend the work of Roodbergen (2012) to provide a deeper analysis of the data generated in our simulations and by developing tools to estimate optimal zone sizes without requiring to run the simulator again. Compared to the remaining literature, our contributions are stated as follows.

- Perform extensive simulations for different zone sizes used in common manual warehouse settings. Other studies only considered small scale simulations for a reduced number of input factors;
- Analyze the data generated by the simulations to provide a quantitative measure for the influence of factors in the zone sizing solution. Previous studies only provide the improvement of the expected ARL when using different settings for their simulations;
- Train machine learning models to estimate the optimal zone sizes from the factors previously described. This methodology has never been applied to solve the zone sizing problem;
- Compare their applicability regarding attributes such as the quality of the solutions provided and the ease of implementation in practice. They are also compared against

commonly used methods, such as the arbitrary zone sizes previously mentioned and the random storage policy;

- Provide methods that can determine good zone sizes in negligible calculation time, which allows for embedding our methods in other warehouse design methods, for example, for optimizing warehouse layout.

3.3 Multi-block warehouse factors and simulator

The routes followed by pickers to retrieve products ordered are influenced by several factors. Le-Duc and De Koster (2005) and Petersen (1999) summarize them to the layout of the warehouse, the demand pattern, the storage strategy, the batching method, and the routing method. Each of these features are discussed next.

3.3.1 Layout factors

In a standard multi-block warehouse, there are five layout factors to be considered: number of aisles, number of cross aisles, aisle length, aisle width, and cross aisle width.

The number of aisles influences travel distances since more aisles containing picks increase the distance to be travelled in a route. Longer aisles may also increase the route length since pickers spend more time travelling in a single aisle before a new one can be entered. Usually, warehouses have at least two cross aisles, one at the front and one at the back of the storage area. Additional cross aisles may reduce route length depending on the pick locations. Finally, wider aisles and cross aisles also increase the travel distance. We ignore the distance between the left and the right side racks in an aisle and consider that pickers walk in the middle of aisles and cross aisles. The total storage capacity of the warehouse is twice the number of aisles multiplied by the aisle length. This is a continuous representation of storage capacity, which generalizes the representation of capacity in terms of total number of slots.

3.3.2 Demand distribution

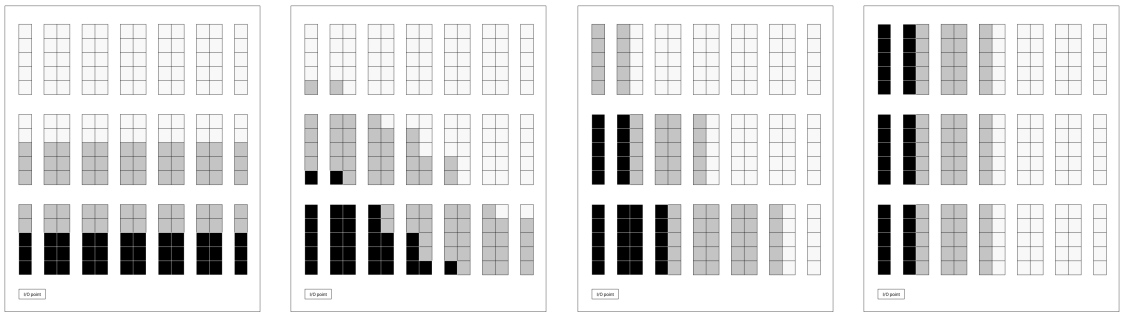
The only factor related to the demand is its distribution. Typically, few products account for the majority of the demand volume. Some related works (Le-Duc and De Koster, 2005; Petersen et al., 2004) consider that the demands are represented by a step function where a certain percentage of products accounts for a certain percentage of the total demand, usually 20% of products accounting for 80% of demand. Caron et al. (1998) present an analytical function to describe this characteristic using a continuous function known as ABC curve. Namely,

$$F(x) = \frac{(1+s)x}{s+x}, \quad 0 \leq x \leq 1, s \geq 0, s+x \neq 0, \tag{3.1}$$

where x indicates the zone size corresponding to the items whose order frequency represent a fraction $F(x)$ of total warehouse activity. The parameter s indicates the skewness of the demand. For example, for $s = 0.067$ it holds that 80% of the picks are generated by 20% of the products, reducing to 70% for $s = 0.12$, 60% for $s = 0.2$ and 50% for $s = 0.333$, which are common skewness values found in the step functions of related works.

3.3.3 Storage strategy

The storage strategy accounts for assigning products within the storage area. The zone size is expected to significantly influence the efficiency of the storage location assignment (Van Gils et al., 2018b). In this work, we consider an ABC storage system. While our main objective is to estimate the optimal combination of zone sizes, the zone shapes are determined following one of the four methods for the ABC storage shown in Figure 3.2. In this figure, different colors define the zones, with the class A being represented in black, class B in grey, and class C in white. *Across-aisle* (Aa) locates the A products in the front-most locations of each pick aisle. *Nearest-location* (Nl) ranks storage locations by their distance to the I/O point and, then, locates the A products in the closest locations to the I/O. Nl is closely related to the method of diagonal storage (Petersen, 1999), which defines boundaries following a diagonal shape. For single product orders, Nl minimizes the expected ARL. *Nearest-subaisle* (Ns) ranks subaisles according to the distance of their heads to the I/O point and, then, the A products are assigned to the closest subaisles. In case more than one zone is assigned to the same subaisle, the products of the best class are assigned to the best locations within this subaisle first. Finally, *within-aisle* (Wa) ranks aisles according to their distance to the I/O point and assigns the A products to the best aisles. Different studies show that each of these policies perform well under different warehouse settings and all of them can actually be found in real cases (Jarvis and McDowell, 1991; Le-Duc and De Koster, 2005; Petersen, 1999; Roodbergen, 2012).



(a) Across-aisle (Aa) (b) Nearest-location (Nl) (c) Nearest-subaisle (Ns) (d) Within-aisle (Wa)

Figure 3.2 – Storage policies

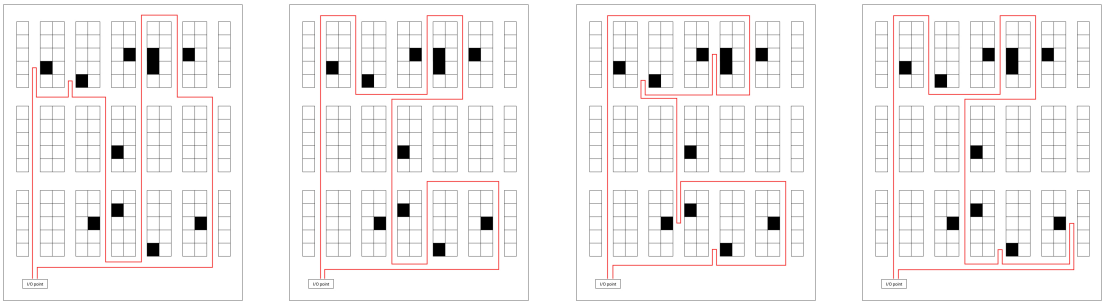
3.3.4 Batching policy

Batching accounts for how the demanded products are grouped to be retrieved by a single picking tour. Several batching policies exist (De Koster et al., 2007). Even though we do not explicitly consider batching, our methodology takes the number of *picks per route*, i.e., the number of locations to visit in a single route, as an input, which may be based on individual orders (pick-by-order) or be the result of a batching procedure. As such, our methods can also be deployed in conjunction with batching procedures for joint optimizations.

3.3.5 Routing policy

Given a pick list and the location of the products contained in it, the order-picking problem (OPP) consists of determining the route to be followed by the picker in order to retrieve them with the objective of minimizing the total distance traveled (Pansart et al., 2018; Scholz et al., 2016). Although exact methods can be found to solve this problem (Pansart et al., 2018; Ratliff and Rosenthal, 1983; Scholz and Wäscher, 2017; Theys et al., 2010), the usage of these methods is intractable, considering that calculation time increase rapidly for an increasing number of cross aisles, and considering the extremely large number of routes that needs to be calculated for our approach. Furthermore, simple heuristic policies are also more likely to be accepted by pickers since they are more intuitive (Grosse et al., 2016).

We consider the four routing policies shown in Figure 3.3. *Aisle-by-aisle* (Aba) considers that each aisle containing at least one pick is visited once. The best cross aisle to move to the next aisle is determined using dynamic programming. In *S-shape* (Sh), pickers traverse the left-most aisle that contains picks to the back of the warehouse and then return to the front picking products one block at a time. Any subaisle containing a pick is traversed. After the last pick in a block, the picker returns to the front of that block and continues to the next. *Largest gap* (Lg) also starts with the picker moving to the back of the warehouse, then picking products one block at a time. Whenever a subaisle contains a pick, instead of entirely crossing it, the picker avoids the "largest gap" and returns to leave it from the same side entered. A gap represents the distance between two adjacent picks, or between the middle of a cross aisle and the nearest pick. The last subaisle of a block is traversed entirely to allow the picker to enter the subaisles from the other side of the block. The last considered policy is the *combined* (Co). It follows the same logic of performing all picks from the back of the warehouse to the front, block by block. However, whenever a subaisle is entered, the picker can choose between traversing it or making a return when all picks within it are done. The choice is made using dynamic programming by always looking one subaisle ahead in order to be in a better starting point for the next subaisle. A detailed description of these policies is found in Roodbergen and De Koster (2001).



(a) Aisle-by-aisle (Aba) (b) S-shape (Sh) (c) Largest gap (Lg) (d) Combined (Co)

Figure 3.3 – Routing policies

3.3.6 Warehouse simulator

Given the five layout factors, the skewness parameter, the number of picks per route, the combination of S/R policies, and the zone sizes, we use a simulator to estimate the expected ARL. The estimations could be done using analytical formulas in some specific situations (Aldarondo and Bozer, 2020; Battini et al., 2015; Hwang et al., 2004; Le-Duc and De Koster, 2005; Ozden et al., 2020; Parikh and Meller, 2010). Notably, they exist for layouts with only one or two blocks, and for specific policies that may differ from those considered here. Since analytical formulas are not available for all situations, our simulator is useful due to its flexibility for accepting any layout and policies among those considered. Furthermore, most analytical formulas provide only approximations of the ARL, and hence using different formulas from different sources may create unintended inconsistencies between data points, which is prevented by using our simulator for all situations. Regression models to predict the best zone sizes can be developed regardless of whether the training data is generated using the simulator or analytical formulas.

In the simulator, the warehouse layout is created first, then the zones are set considering the shapes and locations determined by the storage policy used and their sizes. Then, pick lists are sampled using the ABC curve. They basically contain the zones where each pick will be performed. Next, a random location within that zone is chosen for each pick. The simulator computes the route traveled by the picker to retrieve all products demanded following the routing policy used. This procedure is repeated several times. Each time the route length is found for a new pick list sampled, the estimated ARL and its two-sided confidence interval C are updated as:

$$C_{\alpha/2} = \bar{x} \pm z_{\alpha/2} \frac{s}{\sqrt{n}}, \tag{3.2}$$

where $\bar{x}$ is the estimated ARL, s is the standard deviation of the route lengths estimated from the n pick lists sampled, and $z_{\alpha/2}$ is a value that follows from the cumulative normal distribution function for a chosen degree of certainty $(1 - \alpha/2)$. The simulations stop when

the maximum accepted half-width ϵ is attained, i.e., when (3.3) holds.

$$\epsilon \geq z_{\alpha/2}\frac{s}{\bar{x}\sqrt{n}} \tag{3.3}$$

3.4 Machine learning regression methods

In the context of regression analysis, ML methods are used to predict a set of continuous output values (targets) from a set of input variables (features). These methods are trained using real or synthetic data in order to find a model that best fits the dataset that they are "learning" by finding the correlations between features and targets.

The use of ML regression methods in the warehousing literature is very limited, with most applications found in demand forecasting (Nikolopoulos et al., 2016; Shi et al., 2020), but also in rental price estimation (Ma et al., 2018), forklifts engagement predictions (Mirčetić et al., 2016), and the development of a dynamic routing system for automated guided vehicles (Nguyen Duc et al., 2020).

We test several ML regression methods to estimate the ABC zone sizes that minimize the ARL. The objective is to derive a model that learns the unknown function used by the simulator indirectly leading to the optimal zone sizes through the minimization of the ARL. If we were fitting a model to predict the length of a single route, the ML methods would be learning the set of instructions defined by the routing policy used. However, to predict the ARL, it is reasonable to assume that the unknown function is also dependent on the storage policy used. This means that 16 different functions are to be learned considering the combinations of the four storage strategies (S) of Figure 3.2 combined with the four routing policies (R) of Figure 3.3. For this reason, each of the ML methods used are trained for each combination of S/R policies separately. The dataset used to train them is generated using the simulator considering its input factors (see Section 3.5.3) as features and the best zone sizes as targets.

The ML regression methods used in this study are: ordinary least squares (OLS), regression tree (RT), random forest (RF), and multilayer perceptron (MLP).

In summary, OLS constructs linear functions combining the features to predict each target (Weisberg, 2013). RT is a multi output model composed of several binary decisions performed in a tree-like structure that is able to capture nonlinear relationships between the features and the targets (Loh, 2014). RF is an ensemble method where multiple RTs are randomly created to perform independent predictions, then all predictions are averaged, thus increasing the model robustness (Breiman, 2001). Finally, MLP is a feedforward deep neural network, consisting of an input layer that receives the features, one or more hidden layers where the values of the features are transformed using nonlinear functions, and an output layer that gives the predicted target values (Aggarwal, 2018). These methods represent four of the most commonly used categories of supervised learning methods for regression analysis, i.e.,

linear regression, decision trees, ensemble methods, and neural networks. Other methods were also tested in a preliminary round of experiments, such as ridge and lasso regressions, AdaBoost, gradient boosting, and support vector machines, but only the most promising ones were selected.

3.4.1 Ordinary least squares

OLS is one of the most popular statistical methods used for regression analysis. It fits a linear function using the values of the features to predict the targets. The linear model is such that the sum of the squared deviation is minimized.

Linear regression models assume that target values are unbounded, i.e., they can assume any real number. In the case of ABC zone sizes, they are bounded by zero and the maximum number of storage locations in a warehouse. Multivariate fractional regression models (Murteira and Ramalho, 2016) can be an alternative to use in this situation, however, these models are more appropriate when there are many observations at the upper and/or lower bounds. For ABC zone sizing, it is known that optimal zone sizes lie not too close to their bounds. Otherwise, it would be more advantageous to reduce the number of classes in the solution. In this case, we ignore the bounded nature of the targets and use a regular OLS regression.

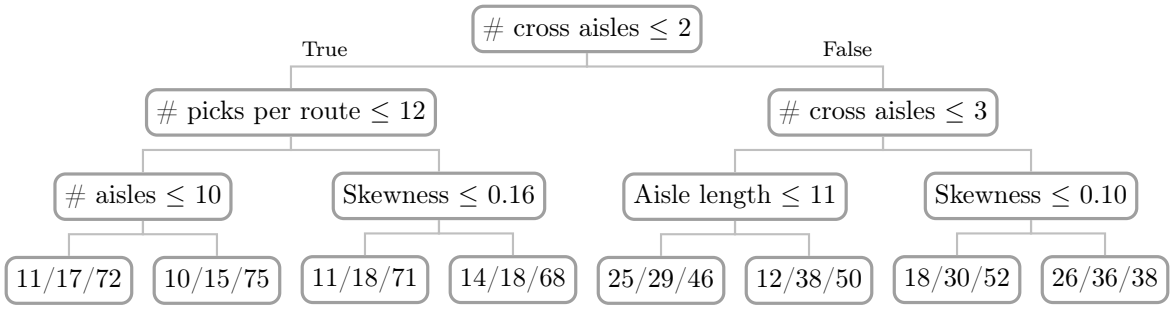
Since OLS supports the prediction of only one target, for the ABC zone sizing, three linear functions are modeled, one for each zone size. Advantages of OLS compared to the other models used here are that it provides simple functions that are easy to apply in real cases and to interpret, since they show which features contribute the most to the predicted values. However, OLS usually does not perform well when the relationships between features and targets are nonlinear.

3.4.2 Regression tree

RT is a non-parametric method that can be used for regression analysis. It is based on a hierarchical decision scheme using a tree like structure, also known as decision tree, as the example in Figure 3.4. The tree contains a root node containing all data, a set of internal nodes and a set of terminal nodes (leaves). At each node, a binary decision is made using a condition associated to one of the input features, until a leaf is reached containing a combination of the estimated target values (Loh, 2014).

Construction of an RT starts from the root node, where its prediction is made based on the target values of the training samples. Then, it searches over all features for a split that minimizes the sum of squared errors for the two new nodes generated. This process continues for each new node generated until a user-defined stopping criteria is reached. An RT is easy to understand and to interpret, since it breaks complex decisions into several simpler ones, hence

Figure 3.4 – Example of an RT for the zone sizing problem

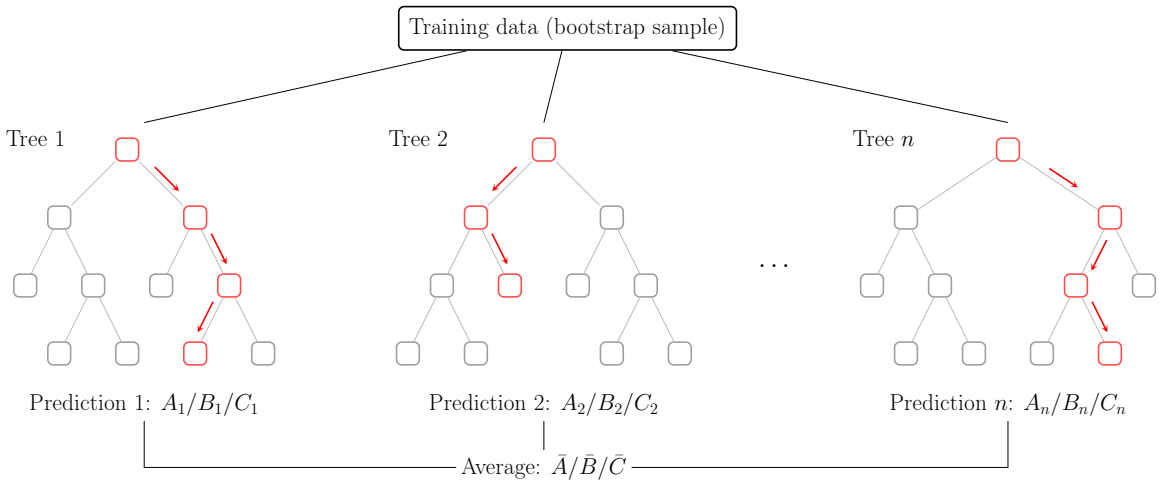


it can be visualized. Moreover, it can be used to deal with complex nonlinear relationships. More on RT is found in Loh (2014).

3.4.3 Random forest

RF belongs to the class of the ensemble methods in machine learning (Breiman, 2001). In ensemble learning, several, usually simple, estimators are combined such that each provides a prediction and, then, all predictions are combined, improving the robustness over single estimators. RF consider multiple RTs, like the one previously shown in Figure 3.4, such that each tree is built from a set of randomly sampled data with randomly sampled features with the same distribution for all trees in the forest (bootstrap sample). The predictions of each tree are averaged for the final solution. The structure of an RF to the ABC zone sizing is shown in Figure 3.5. As the number of trees increases, the error for the forest tends to converge, decreasing the variance of the RF model, and increasing accuracy when applied to unseen data, i.e., a lower overfitting. Due to this reason, RF is robust with respect to outliers.

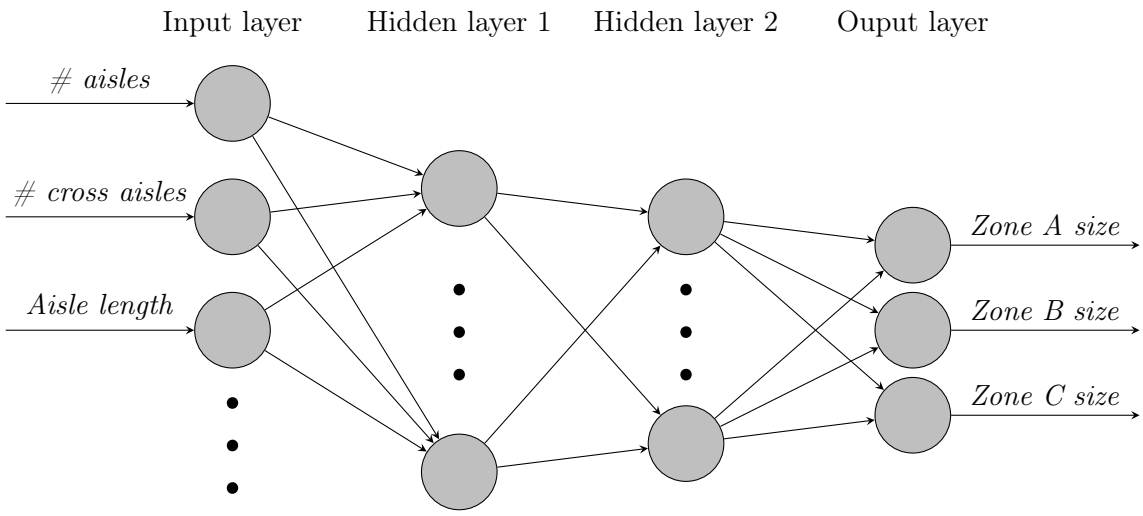
Figure 3.5 – Structure of an RF for the zone sizing problem



3.4.4 Multilayer perceptron

Artificial neural networks are machine learning methods inspired by how the human brain processes information and are used to approximate complex functions including nonlinear relationships that depend on several features. MLP is an artificial neural network composed of a structured network of one input layer, one or multiple hidden layers, and one output layer, as shown in Figure 3.6. The input layer consists of one node for each feature considered, while the output layer consists of one node for each target to be predicted. Each hidden layer is composed of a number of nodes where the values from the previous (input or hidden) layer are input, transformed using a nonlinear activation function, and sent to the next (hidden or output) layer in the forward direction (feedforward architecture). A loss function, usually minimizing the sum of squared errors in regression problems, is optimized in the output layer via backpropagation (Aggarwal, 2018).

Figure 3.6 – Structure of an MLP to the zone sizing problem



MLPs are scale sensitive; if one input feature has a larger range than another, and both have a similar variance, then the MLP has a tendency to be more sensitive to the larger one. Therefore, feature scaling is recommended when setting an MLP (Aggarwal, 2018). Standardization is a common way to scale features and is done as:

$$\hat{X} = \frac{X - \bar{X}}{\sigma}, \tag{3.4}$$

where X represents the feature samples for a feature with mean $\bar{X}$ and standard deviation σ, transforming all features to have a mean of zero and unit variance.

The MLP has been demonstrated to be a very powerful tool since it does not require a high level of abstraction about the data domain and it self-organizes its complexity by adding or

removing neurons according to the available data or computational power (Aggarwal, 2018). Its hidden layers have a non-convex loss function, such that different initializations can lead to different validation accuracy. We detail in Section 3.6.1 how we set the number of hidden layers and the number of neurons in our MLP model.

3.4.5 Model evaluation

The objective of all models is to estimate the ABC zone sizes such that the ARL is minimized. Due to the long time required to run simulations with a high level of confidence, we do not use this metric directly when setting up the models. Instead, we use a common, and fast to compute, score metric for evaluating the model's performance: Mean Squared Error (MSE). MSE measures the variance of the predictions, therefore reducing worst case estimations. The formula for MSE is shown in equation (3.5), where n is the sample size, Y_i is the i-th observed value for the feature and $\hat{Y}_i$ is the value predicted.

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(Y_i - \hat{Y}_i)^2. \tag{3.5}$$

A model is considered to be better for a combination of parameters that result in a lower MSE. We show in Section 3.6 that this metric predicts well the model performance for the original objective of minimizing the ARL.

3.5 Data analysis

In this section, we detail how the simulator was setup to generate the dataset used to train the regression models. We also present an analysis of the relevant information extracted from the data generated. The simulator was implemented in C++ and simulations were run on a parallel cluster of machines with an Intel Gold 6148 Skylake with 2.4 GHz at each node with up to 1,000 nodes allowed to be used in parallel.

3.5.1 Dataset generation

The dataset used to train our models was generated using the warehouse simulator described in Section 3.3. We generate 1000 samples to represent different warehouse settings. Each sample is generated using common values found in practice for the warehouse factors passed to the simulator (Roodbergen, 2012). The values of each factor were selected using a uniform distribution considering lower and upper bounds. This way the samples are generated randomly instead of in a grid pattern, which would require the generation of many more samples to be representative. All values generated are integer, except for the skewness. The bounds used are shown in Table 3.1 together with the S/R policies. Therefore, a total of 16,000 instances were generated.

Table 3.1 – Bounds of the input features

Feature	*Bounds*
Number of aisles	$[6, 18]$
Number of cross aisles	$[2, 6]$
Aisle length (in meters)	$[10, 30]$
Aisle width (in meters)	$[2, 6]$
Cross aisle width (in meters)	$[2, 6]$
Picks per route	$[5, 25]$
Skewness	$[0.05, 0.35]$
Storage policy	{Aa, Nl, Ns, Wa}
Routing policy	{Aba, Sh, Lg, Co}

For each instance, we estimate the ARL by performing a grid search, simulating the combinations of zone sizes $A/B/C$, where $A = \{1\%, 2\%, \ldots, 75\%\}$, $B = \{1\%, 2\%, \ldots, 75\%\}$, and $C = 100\% - A - B$. This results in 4350 feasible combinations for each instance. Pick lists were generated until $\epsilon = 0.25\%$ for a 99% of degree of certainty. The result is a 5 GB data file containing the expected ARL for each combination of zone sizes and instances. Total simulation time took around 12,000 CPU hours, which was only possible to run due to the parallel cluster of machines available.

3.5.2 Target analysis

We first analyze the data obtained from the simulator regarding the best combinations of ABC zone sizes, i.e., the zone sizes that led to the lowest expected ARL for each warehouse setting sample. We will refer to these as the best known solutions (BKS), although due to the inherent uncertainties involved in the simulation of the expected ARL other zone sizes simulated may perform better. A summary of this data is presented in Table 3.2. It shows that the overall BKS found are $A = 18.33\%$, $B = 35.49\%$, and $C = 46.18\%$, which slightly deviates from the commonly used sizes of 20/30/50.

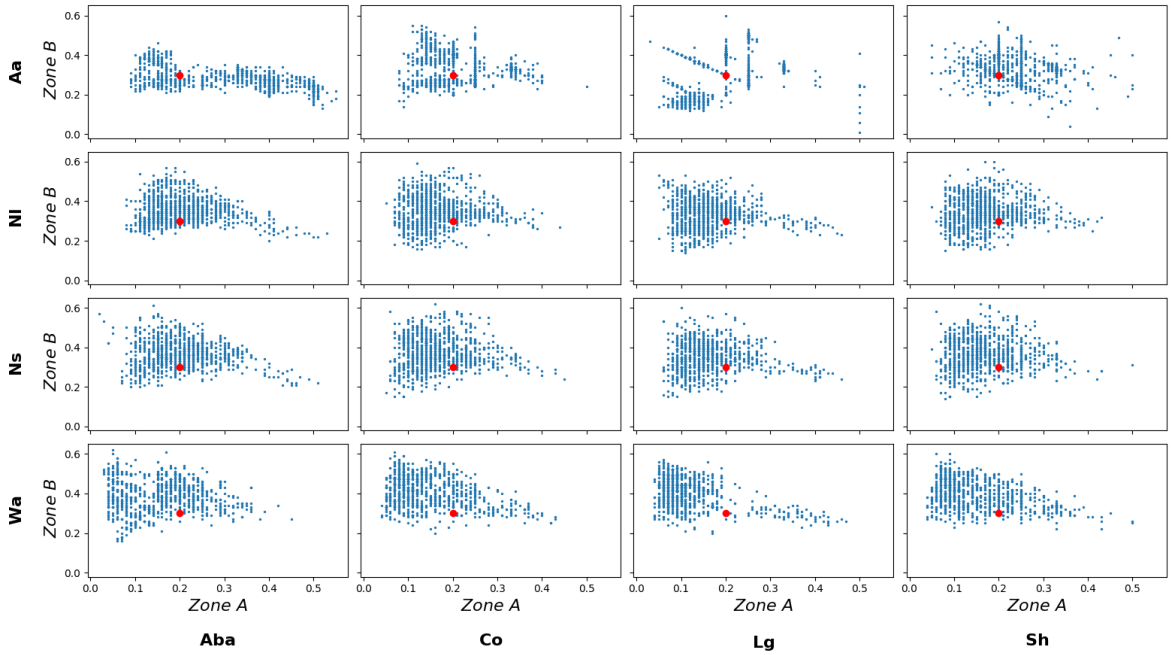
Table 3.2 – Statistics for the best zone sizes found in the simulations

Attributes	*Zone A size*	*Zone B size*	*Zone C size*
Mean	18.33	35.49	46.18
Standard deviation	8.77	8.07	10.22
Minimum	2	1	4
Lower quartile	12	30	39
Median	17	35	46
Upper quartile	23	41	52
Maximum	55	62	81

In Figure 3.7 we plot every BKS found. The x-axis shows A and the y-axis shows B. Red dots indicate the popular 20/30/50 solution. BKS are presented for different combinations of S/R

policies. This figure represents well the high standard deviation and the large range between minimum and maximum observed sizes from Table 3.2, indicating that the use of fixed zone sizes as a rule-of-thumb usually leads to a solution numerically far from the optimal one. Only half of the BKS fall within a range of $A = 12\%$ to 23%, $B = 30\%$ to 41%, and $C = 39\%$ to 52%.

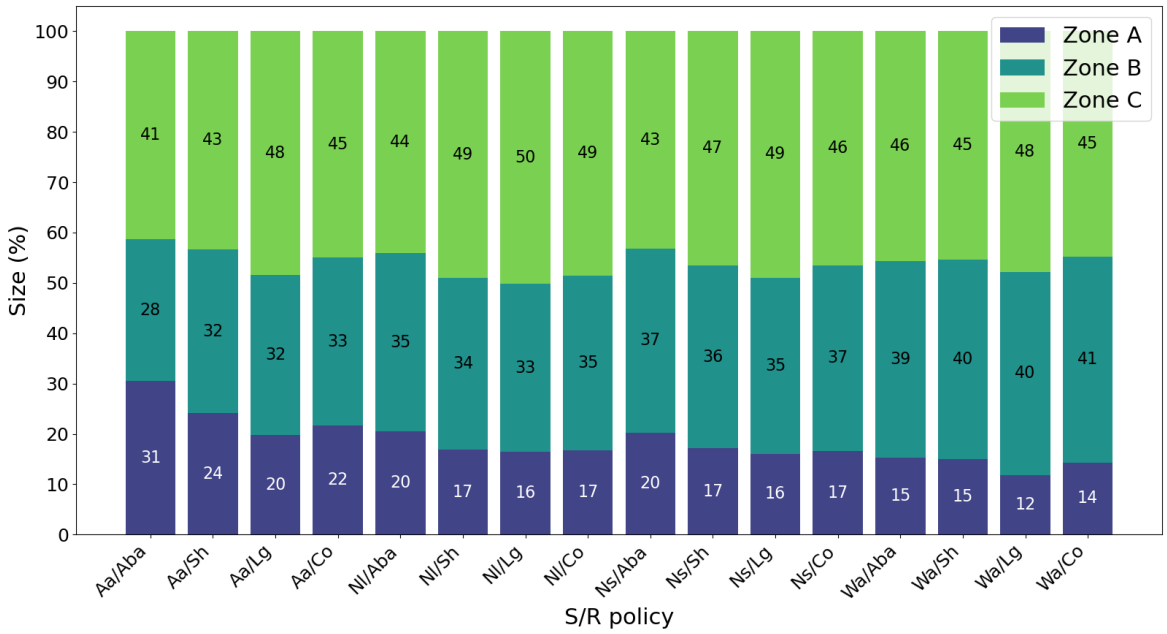
Figure 3.7 – Best zone sizes found in each simulation



We observe different data patterns for different combinations of S/R policies in Figure 3.7. One of them is the diverse density of points for different combinations of zone sizes in each S/R policy. For instance, Wa/Lg has most of the BKS with B above the reference point, while in Aa/Aba the opposite is seen. Figure 3.8 exhibits the mean BKS for each S/R policy. Note that the overall optimal zone A size is more than 2.5 times higher when using Aa/Aba, than when using Wa/Lg. We remind that the figure conceals actual variation in performance within each bar. However, these sizes are expected to lead to a better performance than the common 20/30/50 and can be easily adopted in practice. Later in this paper (see Figure 3.10), we present the gap between the ARL when using these average BKS for each S/R policy and the ARL of the BKS found in the simulations for each sample. We show that although the average performance is better than 20/30/50, the ARL deviation to that of the BKS is still high.

Another metric used to justify the need for having accurate models to predict optimal zone sizes is the number of combinations of zone sizes simulated within a gap δ from the ARL of the BKS. For $\delta = 0\%$, we have approximately one combination per sample, since ties rarely occur. Due to the uncertainty when estimating ARL using the simulator, defined at $\epsilon = 0.25\%$, we

Figure 3.8 – Average best zone sizes per S/R policies found in the simulations



are interested in finding a combination that leads to an ARL below twice the value of ϵ from the best solution. If this is the case, we cannot reject the hypothesis that the solution found is equal to the BKS.

Some combinations of S/R policies are naturally easier to estimate good zone sizes since their performance do not change significantly when the zone sizes are changed marginally. It is the case of Aa/Sh, which places high demanded products in as many aisles as possible with a routing policy that prefers to travel a few aisles entirely. This is a poor combination in practice since bringing highly demanded products to the front of the aisles does not lead to a reduction in the number of aisles visited. The opposite is expected for Wa/Sh, in which placing a single highly demanded product in a new aisle can significantly increase the ARL. This implies that a model to predict optimal zone sizes for the Aa/Sh policies is not required to be as accurate as a model for the Wa/Lg policies to generate good solutions. For this matter, different models are applied for each S/R policy combination.

3.5.3 Features analysis

Feature engineering is a process of using domain knowledge to create/extract new features from a given dataset (Turner et al., 1999). We extract new meaningful features in the warehousing context from the original warehouse factors using nonlinear operations. They are:

- *Number of subaisles*: number of aisles / (number of cross aisles − 1)
- *Subaisle length*: aisle length / (number of cross aisles − 1)

- *Warehouse length*: aisle length + cross aisle width × number of cross aisles
- *Warehouse width*: number of aisles × aisle width
- *Dimensions ratio*: warehouse length / warehouse width
- *Picks per aisle*: picks per route / number of aisles
- *Picks per subaisle*: picks per route / number of subaisles
- *Pick density*: picks per route / (aisle length × number of aisles)

These eight extracted features and the seven original ones constitute the preliminary set of input features used in the models.

We performed univariate linear regressions between each of the 15 features and each of the three targets for each S/R policy. The obtained coefficient of determination (R^2) values are presented in Table 3.3 with those above 0.1 highlighted. Observe that *skewness* (7) has the highest R^2 values in most policy combinations, which means that this feature shares a higher percentage of the variance with the targets than the others. When Nl or Ns storage policies are used, *skewness* is almost always the only feature with a relatively high R^2. For the Aa policy – except when used with Sh – *number of cross aisles* (2), *number of subaisles* (8), *subaisle length* (9) and *picks per subaisle* (14) are the most linearly correlated with the best zone sizes found. Note that all these features are related to each other since (8), (9) and (14) indirectly contain (2). Finally, when using Wa – except with Lg – several features have a relatively high R^2. We highlight that none of the correlations observed are absolutely high (above 0.5), which is an indication that either the features are not correlated at all with the targets or they have a nonlinear correlation.

3.6 Models setup

In this section, we present how the four regression models are set to predict optimal ABC zone sizes. Each model has its most relevant parameters set using the data generated in the simulations, then the performances of each model with the best settings are compared. Finally, we present an additional step to improve model performance based on the data previously analyzed.

The models were implemented using the Python package Scikit-learn (Pedregosa et al., 2011). We demonstrate the easiness to set the models by performing all training and testing on a personal computer with a six-core Intel Core i5-9400 with 2.9 GHz. The resulting equations of our OLS and the RT are made available online on `https://www.leandro-coelho.com/warehousing-zone-sizes/` to enable a visualization and to facilitate the use of our results.

Table 3.3 – R^2 for the univariate linear regression between warehouse factors and zone sizes

S/R policy	*(1)*	*(2)*	*(3)*	*(4)*	*(5)*	*(6)*	*(7)*	*(8)*	*(9)*	*(10)*	*(11)*	*(12)*	*(13)*	*(14)*	*(15)*
Aa/Aba	0.00	**0.25**	0.00	0.00	0.01	0.01	**0.20**	**0.14**	**0.25**	0.05	0.00	0.02	0.01	**0.10**	0.02
Aa/Sh	0.01	0.04	0.01	0.00	0.01	0.01	0.05	0.01	0.04	0.03	0.01	0.03	0.00	0.02	0.00
Aa/Lg	0.00	**0.33**	0.00	0.00	0.00	0.02	0.03	**0.19**	**0.37**	**0.10**	0.00	0.03	0.00	**0.22**	0.00
Aa/Co	0.01	**0.16**	0.00	0.01	0.01	0.01	0.08	0.08	**0.19**	0.04	0.01	0.03	0.00	**0.10**	0.00
Nl/Aba	0.03	0.04	0.03	0.05	0.01	0.00	**0.37**	0.03	0.04	0.01	0.08	0.08	0.01	0.02	0.01
Nl/Sh	0.00	0.02	0.02	0.02	0.00	0.00	**0.24**	0.01	0.02	0.02	0.03	0.01	0.00	0.04	0.01
Nl/Lg	0.02	0.02	0.03	0.05	0.00	0.01	**0.21**	0.03	0.06	0.00	0.08	0.03	0.01	0.04	0.00
Nl/Co	0.01	0.03	0.01	0.03	0.01	0.00	**0.23**	0.02	0.02	0.02	0.04	0.02	0.00	0.04	0.00
Ns/Aba	0.02	0.01	0.02	0.03	0.00	0.02	**0.29**	0.00	0.01	0.02	0.05	0.04	0.02	0.01	0.03
Ns/Sh	0.01	0.07	0.01	0.02	0.01	0.02	**0.14**	0.05	0.08	0.03	0.02	0.01	0.02	**0.13**	0.02
Ns/Lg	0.02	0.03	0.02	0.04	0.01	0.01	**0.23**	0.04	0.04	0.02	0.06	0.03	0.02	0.02	0.00
Ns/Co	0.01	0.02	0.01	0.01	0.01	0.01	**0.18**	0.02	0.03	0.02	0.01	0.00	0.01	0.06	0.02
Wa/Aba	0.07	0.07	0.01	0.06	0.01	0.06	**0.18**	0.06	0.05	0.06	**0.10**	**0.14**	0.06	**0.11**	0.04
Wa/Sh	0.01	**0.13**	0.01	0.03	0.00	0.04	**0.19**	**0.10**	**0.13**	0.05	0.03	0.03	0.03	**0.17**	0.03
Wa/Lg	0.00	0.02	0.01	0.03	0.01	0.05	**0.25**	0.01	0.03	0.01	0.03	0.02	0.01	0.01	0.02
Wa/Co	0.02	**0.10**	0.02	0.04	0.00	0.05	**0.21**	0.09	0.08	0.06	0.02	0.04	0.03	**0.11**	0.04

*(1): Number of aisles, (2): Number of cross aisles, (3): Aisle length, (4): Aisle width, (5): Cross aisle width, (6): Picks per route, (7): Skewness, (8): Number of subaisles, (9): Subaisle length, (10): Warehouse length, (11): Warehouse width, (12): Dimensions ratio, (13): Picks per aisle, (14): Picks per subaisle, (15): Pick density

For all models, we split the 1000 sampled warehouses arbitrarily into 67% as a training set and 33% as a test set. For the training set, we filter the solutions that result in an ARL gap up to δ percent to the ARL of the BKS found in the simulations. From preliminary experiments, we observed that OLS, RT and RF performed better when fit to a training set that contains only data with $\delta = 0\%$, such that only the BKS are used for training. We present later in Section 3.6.1 how δ was determined for the MLP. For the test set, we only use the BKS, since we want to compare the predicted values with the best ones.

3.6.1 Models development

The regression models are setup after the execution of a dimensionality reduction to remove unnecessary features and a parameter setting. These steps are presented next.

Dimensionality reduction

The addition of highly correlated features may lead to some undesirable conditions, such as overfitting, overly complex models, and more computational time to fit. Dimensionality reduction is the process of transforming a dataset such that only the relevant features are used. We use the backward search to observe which features can be removed from the linear (OLS) and nonlinear (RT, RF, MLP) models in order to improve their performance. Backward search iteratively removes one feature at a time considering the best removal strategy, i.e., the feature that degrades the least the model performance (Marill and Green, 1963). For OLS and RT, fewer features lead to cleaner models, which are easier to interpret and implement in practice. From the experiments, we observe that the use of the same features selected for RT in the other nonlinear models leads to a better performance, besides being trained faster. The

results of the backward search for the linear and nonlinear models with default parameters used in Scikit-learn are presented in Table 3.4. It shows the feature removed in each iteration of the search, and the MSE values for all models fit for both training and test sets. Since no results are obtained after removing the last feature, nothing is shown in the last row of the table. In preliminary experiments we have also used other search procedures, such as forward search, but no significant difference in the features selected was observed.

Table 3.4 – Results for the backward search feature selection method (which feature is removed next)

OLS			*RT*		
Feature	*MSE train*	*MSE test*	*Feature*	*MSE train*	*MSE test*
All features	36.27	43.63	All features	1.03	45.38
Pick density	36.42	37.66	Pick density	1.03	38.46
Cross aisle width	36.51	**37.66**	Cross aisle width	1.03	38.77
Warehouse width	36.69	37.69	Warehouse width	1.03	38.34
Number of subaisles	37.00	37.96	Number of subaisles	1.03	38.46
Aisle length	37.59	38.35	Picks per subaisle	1.03	37.85
Number of cross aisles	38.18	38.89	Subaisle length	1.03	38.01
Picks per aisle	38.93	39.67	Warehouse length	1.03	37.46
Dimensions ratio	39.83	40.48	Aisle length	1.03	36.65
Warehouse length	40.79	41.26	Aisle width	1.03	36.13
Number of aisles	42.51	42.73	Number of aisles	1.03	**36.08**
Aisle width	44.54	44.33	Picks per route	1.03	38.09
Picks per route	46.13	46.02	Picks per aisle	1.13	52.43
Picks per subaisle	47.55	47.60	Dimensions ratio	29.96	47.84
Subaisle length	56.31	55.89	Number of cross aisles	53.03	56.29
Skewness	–	–	Skewness	–	–

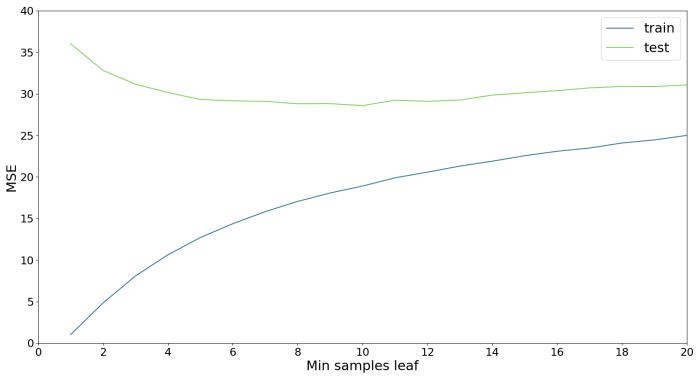
From the results shown in Table 3.4, we observe that the best performance for the test set is obtained when 13 features are used in OLS and 5 features in RT. The most important features are read from the bottom to the top of the table. Unsurprisingly, *skewness* is the most important feature overall for both methods. Another relevant feature is *picks per route*. *Number of cross aisles* and *dimensions ratio* seem to be very relevant in RT, but not so much in OLS, which could be explained by these features having a considerable nonlinear correlation with the targets, but a very low linear one. It is interesting to see that most of the artifical features contain significant information to improve the OLS performance, but for RT some of them are actually used to substitute some of the original features in the models. For example, combining *aisle length*, *number of aisles*, *cross aisle width* and *aisle width* with *number of cross aisles* seems to generate a new feature (*dimensions ratio*) that contains more significant information for the problem than considering the first four as separate features. For the remainder of our experiments, we use OLS with the best 13 features (the last 13 features in Table 3.4). RT, RF and MLP are trained with the best five features (the last five features in Table 3.4). Next, we detail how the parameters of these models were set in this study.

Parameter setting

All methods except for the OLS require some parameter tuning to improve performance. We show next how these parameters were set.

Regression tree. The results for the feature selection presented in Section 3.6.1 show a very high difference in performance between the train and test sets in the RT model. This indicates that when RT is trained using the default parameters, it might result in overfitting. Commonly, pruning is used to reduce overfitting. Methods used to stop early the tree building process are known as pre-pruning, which avoids growing an overly complex tree (Esposito et al., 1997). The pre-pruning technique considered here determines a minimum number of training samples (*min samples leaf*) required to be at a leaf node. So, a node will only be split if it results in two other nodes containing at least this number of samples. The choices for this parameter ranged from 1 to 20. As shown in Figure 3.9, overfitting is observed when the value is too low. As the value increases, the curves representing the MSE for the training and test sets converge. Past the point where MSE for the test set is minimum (*min samples leaf* = 10), both curves start to increase together, leading to an underfitted model. For a better performance, we set the *min samples leaf* parameter to 10 for the remaining experiments.

Figure 3.9 – Results for the setting of the *min samples leaf* parameter in the RT model



Random forest. Random forests depend primarily on two parameters: the number of estimators (n in Figure 3.5) and the maximum number of features for the bootstrapping. The number of estimators represents the number of trees in the ensemble. Overall, more trees reduce variance at the cost of increased computation time. After preliminary tests, we keep the number of estimators at the Scikit-learn default value of 100. The maximum number of features represents the number of features randomly chosen at each estimator. More features make the trees more similar, while fewer features can make trees more diverse. However, too few features may turn the trees too biased. We run another batch of experiments to set the maximum number of features, setting it from one to all five features chosen using the backward search for the RT. The results show that the MSE for the test set is at its lowest point (MSE = 19.88) when the maximum number of features is four. We use this value for the remaining

experiments using RF.

Multilayer perceptron. The parameters to tune in the MLP are the number of hidden layers and the number of neurons in each hidden layer. Before tuning them, we performed experiments for different δ values to observe how MLP behaves when more information about the gaps are used. The parameter δ determines which data from those generated by the simulator are used to train a model. A higher δ can help the model learn more about the unknown function to compute the ARL inside the simulator. However, it also means significantly more data are used to train, which may lead to an exponential increase in the training time. We train the RF with default parameters for $\delta = 0.0\%$ to 0.5%. For that, we have added a new feature to the dataset called *gap*. This feature measures the distance of the observed data point to the best solution obtained for the same problem. This way, *gap* in the training set varies between zero and δ, while for the test set *gap* is always zero, since we are interested in predicting the optimal solution. Table 3.5 shows the results obtained for an MLP with a single hidden layer with 10 neurons and the maximum number of iterations set to 5000 to allow convergence in the optimization process. The remaining parameters are as default in Scikit-learn. All features were scaled. We observe that the performance for the test set increases with more data used for training, but peak performance is obtained with $\delta = 0.1$. Following the goal of maximizing performance, we train the MLP in the remaining experiments using $\delta = 0.1$.

Table 3.5 – Results for the setting of the δ parameter in the MLP model

δ	0.0	0.1	0.2	0.3	0.4	0.5
MSE train	32.17	27.78	32.75	38.97	45.45	52.11
MSE test	34.11	**28.03**	28.18	29.65	30.17	29.89

As the results in Table 3.5 show, the performance of the MLP using the parameters described is worse than RF. In order to improve it, the two parameters are set using a grid search. We tested the number of hidden layers from one to three with 20, 50, 100, 200 and 300 neurons each. The remaining MLP parameters are as in the previous round. Table 3.6 shows the results obtained in the grid search. The best setting found is the use of two layers with 200 neurons each. The MSE for the test set improved from 28.03 to 19.63 when compared to using a single layer with 10 neurons.

3.6.2 Models testing

In this section, we compare the performance of each model for the test set. First, we recap how each model performed using the MSE metric and the training time (Table 3.7). The results show that OLS scores worse than the other models for the test set, while MLP generates solutions closer to the best solutions found in the simulations. However, the time to fit the MLP is significantly higher than the time to fit the RF, and the score improvement is marginal.

Table 3.6 – Results for the setting of the number of hidden layers and neurons parameters in the MLP model

Neurons per layer	*1 layer*		*2 layers*		*3 layers*	
	MSE train	MSE test	MSE train	MSE test	MSE train	MSE test
20	24.01	25.26	19.96	22.14	18.10	21.37
50	20.62	22.81	16.45	20.30	15.06	20.40
100	18.59	21.26	15.43	19.92	13.67	19.88
200	17.81	20.77	15.06	**19.63**	13.33	20.03
300	17.37	21.08	14.61	19.84	12.83	19.77

Table 3.7 – MSE among different models

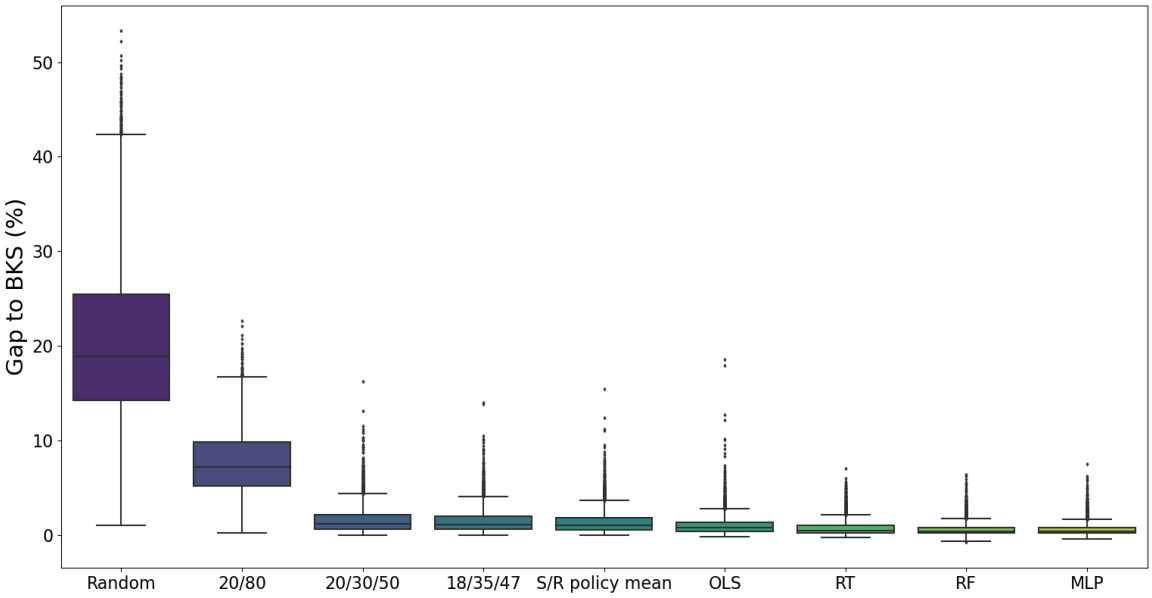
Method	*MSE train*	*MSE test*	*Time to fit (s)*
Ordinary least squares	36.51	37.65	5.8
Regression tree	18.90	28.58	5.8
Random forest	3.60	19.85	8.4
Multilayer perceptron	15.06	19.63	173.1

A model that performs well in the MSE metric will not necessarily do well for the zone sizing problem. In order to measure the real model performances, we run the simulator using the predicted zone sizes for the test set and compared the expected ARL against those from the BKS found by the grid search done using the simulator. In Figure 3.10, we compare the quality of the predictions made by each model against a *random* policy, which is equivalent to the use of a single zone, a *20/80* policy representing the most common partition for a two-zone system, a *20/30/50* policy representing the most common sizes used for an ABC system, a *18/35/47* policy representing the average target values shown in Table 3.2, and *Avg per S/R* representing the average target values derived for each S/R policy shown in Figure 3.8. These results are for the ensemble of the 16 models trained for each combination of S/R policies. In B.1, we detail the average and worst case results for each of them.

The results shown in Figure 3.10 indicate that the use of three zones performs significantly better on average than a random storage or a two-zone system, regardless of the method used to define zone boundaries. This is in accordance with the findings of Eynan and Rosenblatt (1994) for a warehouse with single-command cycles and De Koster et al. (2012) for a multiple picking setting with a conveyor belt. Moreover, although 20/30/50 is the most common combination used in practice, 18/35/47 has on average a better performance (1.61% versus 1.50%), and if arbitrary sizes are picked from the list derived for each S/R policy, the solutions are even better on average (1.37%).

Regarding the first regression method, although the linear relationship between features and targets is weak, OLS is capable of improving the average gap to the BKS compared to the arbitrary sizes (1.05%), but the variance in performance remains high. RT, which is a com-

Figure 3.10 – Methods gaps to the ARL of the best combination of zone sizes found in the simulations



peting method due to its ease of interpretability and replicability, has an even better average performance (0.76%) and a much lower variance.

As expected, RF and MLP show the best average performances among all methods (0.65% both). Also, variance is low with a slight advantage to MLP. On the other hand, RF finds better solutions in the best cases, sometimes even finding solutions with a negative gap to the BKS.

These conclusions attest that the models that performed better for the MSE metric also performed better for reducing the estimated ARL, even though MSE is not contained in the objective function of the zone sizing problem. Also, regardless of the policies, all average solutions are improved when using a trained ML model.

3.6.3 An improvement step: round to the nearest subaisle

As shown in B.1, variations in performance in all methods are observed between models for each S/R policy. As explained in Section 3.5.2, some S/R policies are naturally easier to find good estimations for the optimal zone sizes. Overall, all models can predict better zone sizes when Aa or Nl are used than when Ns or Wa are used. The latter are storage policies linked to the sizes of aisles and subaisles. We noted that among the good solutions for these two policies are those that zone boundaries overlap the point where aisles or subaisles end. This knowledge is used to improve the predictions of the models by rounding them to the percentage representing the nearest number of full subaisles that each zone should cover. For

example, instead of predicting that A covers 14.7 subaisles, it is rounded such that it contains 15 subaisles. In Table 3.8, we show the new average and maximum ARL gaps to the BKS for each S/R policy using the *round to the nearest subaisle* (RNS) procedure as an additional step for the predictions done by the RF. The average ARL gap of the predicted solutions to the BKS improves from 0.65% to 0.50%, but the gains are most notable in the predictions for the Ns and Wa policies.

Table 3.8 – Comparison between the original predictions by RF against the predictions rounded to the nearest subaisle

S/R Policy	*RF*		*RF + RNS*	
	Avg (%)	Max (%)	Avg (%)	Max (%)
Aa/Aba	**0.27**	**1.42**	0.28	1.71
Aa/Sh	0.29	**5.44**	**0.24**	6.01
Aa/Lg	0.66	**4.74**	**0.65**	5.14
Aa/Co	**0.30**	**2.57**	**0.30**	3.07
Nl/Aba	**0.35**	**2.28**	0.39	2.32
Nl/Sh	**0.46**	**2.74**	0.52	3.76
Nl/Lg	**0.47**	**3.63**	0.50	3.75
Nl/Co	**0.38**	**1.90**	0.43	2.24
Ns/Aba	0.85	5.62	**0.59**	**2.50**
Ns/Sh	1.01	4.18	**0.64**	**3.99**
Ns/Lg	1.06	5.25	**0.71**	**3.39**
Ns/Co	0.91	**4.77**	**0.66**	5.84
Wa/Aba	1.01	**6.36**	**0.79**	6.45
Wa/Sh	0.85	3.85	**0.46**	**3.54**
Wa/Lg	0.69	6.30	**0.39**	**3.44**
Wa/Co	0.80	**4.98**	**0.50**	5.34
Average	0.65	4.13	**0.50**	**3.91**

The additional RNS step led the RF to achieve a remarkable milestone. We remind that the stopping criterion used for the simulations was a maximum half-width $\epsilon = 0.25\%$. In an analysis done in Section 3.5.2, we mentioned that zone sizes that lead to an ARL of up to twice ϵ can also be considered to be optimal since we cannot reject the hypothesis that the BKS is equal to it. After the RNS step, the average gap of the predicted solutions reached this threshold, which means further improvements past this point are statistically meaningless. In fact, 67.9% of the predictions made by RF after rounding are below a 0.5% gap from the BKS. Before rounding, this metric was only at 41.6%. Gains with rounding to the nearest subaisle are observed in all models used for the Ns and Wa storage policies. This is an important information extracted from the data to be considered as a simple guideline for practitioners to improve zone sizing solutions.

3.6.4 Managerial analysis

In this section, we provide a summary of the results obtained in the previous experiments and a discussion to highlight the pros and cons of using each method presented to solve the zone sizing problem in practice.

Summary of the results obtained

We summarize the analysis of the results of the computational experiments done for a quick reference:

- We observed that the arbitrary sizes 18.33/35.49/46.18 (rounded to 18/35/47) represent a better choice overall than the most common choice 20/30/50;
- Given the combination of S/R policy in use, better arbitrary sizes can be chosen with zone A size ranging from 12% to 31%, zone B size ranging from 28% to 41%, and zone C size ranging from 41% to 50% (see Figure 3.8);
- Different S/R policies have different levels of accuracy required to estimate good zone sizes. For example, it is easier to estimate good zone sizes for Aa/Sh than for Wa/Lg;
- For most of the S/R policies, *skewness* is the most relevant feature (see Table 3.3). However, its linear correlation is not high enough to justify its use alone to estimate good zone sizes;
- The exception for the previous statement is when Aa is the storage policy in use. In this case, the features related to the subaisles (number, length, etc.) are the most relevant ones. However, they are not enough to provide good estimations of optimal zone sizes;
- Although 13 features have to be considered together such that OLS can provide good predictions, RT requires only five features – skewness, number of cross aisles, dimensions ratio, picks per aisle, and picks per route. This indicates that the correlation between these features and the zone sizes is nonlinear;
- After setting the models to reduce overfitting, using MSE as evaluation metric indicates that the performance of $OLS < RT < RF = MLP$ (see Table 3.7). The same is observed when we change the evaluation metric to the expected ARL, proving that MSE is an adequate metric to use to train the models;
- All four ML models could provide an overall performance better than the arbitrary zone sizes, with an average reduction of the ARL near 1% and worst case scenario reducing up to 15% when the best models are used;
- Finally, we suggest an additional step of rounding predictions to the nearest subaisle. This led to significant performance improvement for the Ns and Wa policies.

Methods comparison

Whenever developing methods to solve optimization problems, it is usual to give all attention to their performance and overlook their applicability. In practice, there are many variables that

are usually disregarded when modeling the problem. They may play a vital role when deciding whether a better performing method is of practical use. For the zone sizing problem, we present a set of seven important characteristics a method must have in order to be applicable. All methods presented in this work are evaluated according to these attributes. The first one is, obviously, the method's *performance* considering the average quality of the solutions obtained. Another desirable attribute is the *variance* of solutions. Ideally, a method provides similar solutions in terms of quality regardless of the warehouse setting. Infeasible solutions, such as negative zone sizes or sizes that sum up more than 100%, can be given so that the *infallibility* is another attribute to be considered. An infallible method can always provide feasible solutions, disregarding its performance, even for unusual features. The fourth attribute evaluated is the *interpretability*. Methods that result in interpretable models have a decision process easy to understand and, therefore, to be accepted by practitioners. This can reduce the resistance for its adoption, especially when it results in unusual zone boundaries. Another desired attribute is the *quickness* to train and run the method until satisfactory solutions are obtained. Specifically for the zone sizing problem, it is important to evaluate whether a method requires a demand *forecasting accuracy* in order to make good predictions. Some methods are more sensible to the precision of the real skewness feature than others and, therefore, accuracy in the demand forecasting is required. Finally, the implementation *cost* of the method should be considered. The cost can be measured based on the overall effort required for its implementation.

Table 3.9 provides a summary of how each method is classified among these seven attributes. Arbitrary sizes consider zone boundaries established from commonly used reference values, such as the 20/30/50, 18/35/47 or using a reference table such as the one for the S/R policy combinations. For OLS and RT, we consider the adoption of the models provided in this work, which are ready to be adopted and easy to understand since they consist of simple arithmetic operations in the OLS case or a tree with binary decisions in the RT case. For RF and MLP, we understand that the models trained here are hardly replicable. Therefore, new models would have to be trained, for example, using real data from the warehouse operations. Finally, for the use of simulations as a solving method, we consider that the warehouse would have to implement their own tailor-made simulator and design the experiments to search for the best combination of zone sizes, which may require many decisions to be made such as which zone sizes combinations to test, and the desired degree of certainty for the results obtained, which are decisions not required when using the other methods presented here.

From a practical perspective, no single method absolutely outperforms others for all attributes. Overall, arbitrary sizes are better applicable in cases where no demand forecasts are available. This is hardly the case for existing warehouses. The only advantage of OLS over RT is its slightly easier interpretability. However, RT has many more advantages, therefore being more recommended overall. RF and MLP are better options when the slightly better performance they provide can result in significant savings, for example, in companies with large scale

Table 3.9 – Qualitative comparison between different methods to solve the zone sizing problem

Method	*Performance*	*Variance*	*Infallibility*	*Interpretability*	*Quickness*	*Forecasting accuracy*	*Cost*
Arbitrary sizes	Poor	Very high	Yes	Very easy	No run required	None	None
OLS	Fair	High	Can fail	Very easy	No run required	Required	None
RT	Good	Average	Yes	Easy	No run required	Desirable	None
RF	Very good	Average	Yes	Hard	Fast train and test	Desirable	Low*
MLP	Very good	Low	Can fail	Very hard	Slow train, fast test	Required	Low*
Simulator	Excellent	Very low	Yes	Average	Very slow	Required	High

*Considering that data for training is readily available. Otherwise, gathering data for the warehouse performance under different zone sizes can be significantly more expensive than building a simulator

operations and many warehouses. Therefore, it is reasonable to assume that data is available to train them. The simulator is recommended in case an even better performance is desirable and enough resources are available to implement it.

3.7 Conclusions

This paper investigates the ABC zone sizing problem, which arises from the class-based policy used to solve the storage location assignment problem. In this problem, a multi-block warehouse is divided into three zones of sizes to be determined. The set of products to be assigned to storage locations are divided into three classes according to their demands, and, the best classes are assigned to the best zones. Although the zone sizing is a common problem faced in manual warehouses, the literature about it is still scarce.

We have generated detailed synthetic data to represent real warehouses and customer orders using a simulator to estimate the average route length. The simulator is fed with a set of warehouse factors related to its layout, the pick list characteristics, and the storage and routing policies used. An analysis of the data obtained from an extensive batch of simulations revealed that good zone sizes vary significantly within a large range. Only half of the optimal zone sizes for the common warehouse settings analyzed fall within a range of 12% to 23% for zone A, 30% to 41% for zone B, and 39% to 52% for zone C. We have also shown that some combinations of storage and routing policies are harder to approximate good solutions than others. We observed that all considered features have weak linear correlations with the zone sizes, and that the demand skewness is the most correlated feature for most storage and routing policies combinations.

We have then used the data from the simulations to train several machine learning regression methods to learn on how to predict the optimal zone sizes. This was done by showing to them what are the best solutions found for different settings. This approach provides a faster way to solve the problem compared to implementing a simulator, which can be too time-consuming for warehouses. The models that can capture nonlinear relationships between features and targets performed better than the linear regression. We have shown that a set of only five features – demand skewness, number of cross aisles, dimensions ratio, picks per aisle, and

picks per route – is enough to obtain a good performance using these nonlinear models. We have also suggested an additional step of rounding predictions to the nearest subaisle for some policies to improve the predictions.

There is a trade-off between performance and applicability among the models. Arbitrary sizes are easy to remember and significantly outperform alternative methods such as the random storage or a two-zone system. However, they do not fully benefit from the performance potential of using a three-zone system. The linear functions and decision trees are interpretable models that consider the problem features to improve performance. Even though linear correlations are weak, the linear regression model outperformed the more primitive solutions. The decision tree trained predicts solutions on average 0.76% from the best ones found in the simulations and it is a method that can be easily adopted using the trees generated here. For even better results, an ensemble method, such as the random forest, or a neural network, such as the multilayer perceptron, can be used. We used a random forest to show that solutions with an average gap of 0.5% to the best found in the simulations are achievable. At this point, the loss of solution's quality is statistically insignificant, such that most of the solutions predicted are no different than those provided by the simulations.

This study can be extended by considering different characteristics commonly found in manual warehouse, such as the influence of other batching policies or the use of zone picking. The success of the proposed methodology is also an indication that it may work for different problems and warehousing systems, given that enough data is available or can be obtained to train the ML models.

Chapter 4

Quadratic assignment problem variants: a survey and an effective parallel memetic iterated tabu search

Chapter information A paper based on this chapter is published in *European Journal of Operational Research*: A. Silva, L. C. Coelho, and M. Darvish. Quadratic assignment problem variants: a survey and an effective parallel memetic iterated tabu search. European Journal of Operational Research, 292(3):1066–1084, 2021.

Résumé

Dans la politique de stockage corrélée, les produits constamment prélevés ensemble doivent être affectés à des emplacements proches pour réduire la distance moyenne attendue des routes. Cette politique peut être modélisée comme une problème d'affectation quadratique (PAQ). Dans le PAQ, nous minimisons les interactions entre paires de produits. Certaines variantes existent comme lorsque la plus grande interaction entre produits est minimisée (affectation de goulot d'étranglement), lorsque les interactions entre groupes de quatre produits sont minimisées (affectation biquadratique), lorsque le nombre de produits et d'emplacements ne sont pas égaux (semi-affectation quadratique), ou lorsque des multiples produits peuvent être affectés au même emplacement (affectation quadratique généralisée). Dans ce chapitre, nous présentons une métaheuristique itérative de recherche tabou mémétique en parallèle pour résoudre le PAQ et ses variantes. Des expériences utilisant des instances de référence de la littérature attestent de l'efficience de notre métaheuristique, montrant sa compétitivité par rapport aux meilleures méthodes existant, soit séquentielles ou parallèles. Nous montrons que notre métaheuristique surpasse de manière significative les meilleures méthodes trouvées pour les quatre variantes du PAQ, mettant à jour de manière significative leur littérature.

Summary

In the correlated storage policy, products constantly picked together should be assigned to close storage locations to reduce the expected average route length. This policy can be modeled as a quadratic assignment problem (QAP). In the QAP, we seek to minimize the interactions between pairs of products. Some studied variants are when the largest interaction is minimized (bottleneck assignment), when interactions between groups of four products are minimized (biquadratic assignment), when the number of products and locations are not equal (quadratic semi-assignment), or when more than one product can be assigned to the same location (generalized quadratic assignment). In this chapter, we present a powerful parallel iterated tabu search metaheuristic to solve the QAP and the four variants mentioned. Computational experiments using the hardest benchmark instances from the literature attest the effectiveness of our metaheuristic, showing its competitiveness when compared to the state-of-the-art methods, sequential and parallel. We show that our metaheuristic significantly outperforms the best methods found for all four variants of the QAP, significantly updating their literature.

4.1 Introduction

Consider an assignment problem where facilities have to be located in sites. Each facility can be located in one site and each site can hold exactly one facility. The distances between sites and the flows between facilities are given. The Quadratic Assignment Problem (QAP), introduced by Koopmans and Beckmann (1957), consists in minimizing a cost function expressed by the distance and the flow between each pair of facilities assigned to their respective sites. The QAP is a well-known and well-studied problem, being considered as one of the most difficult to solve in combinatorial optimization. Even finding an approximate solution to the QAP is hard due to the quadratic form of its objective function in addition to its combinatorial nature (Sahni and Gonzalez, 1976). The QAP has received considerable attention during the last decades, as shown by the several QAP books and reviews (see Finke et al. (1987), Loiola et al. (2007), Burkard (2013) and Abdel-Basset et al. (2018b)).

Several problems related to the QAP appear in literature. The Quadratic Bottleneck Assignment Problem (QBAP) (Steinberg, 1961) aims to minimize the maximum interaction cost between two facilities instead of the overall network cost. The Biquadratic Assignment Problem (BiQAP) (Lawler, 1963) is essentially a QAP in which the interactions occur between quadruple of facilities simultaneously instead of pairs. The Quadratic Semi-Assignment Problem (QSAP) (Greenberg, 1969) relaxes the constraints that all sites have to be assigned to exactly one facility, allowing the number of potential sites to be different than the number of facilities. A relatively new problem is the Generalized QAP (GQAP) (Lee and Ma, 2004), which considers that multiple facilities can be located in a site, if enough resources are available. Other less used and studied variants also exist (Smith and Li, 2001; Knowles and Corne,

2003; Hahn et al., 2008b, 2010; Punnen and Wang, 2016).

A number of practical problems can be formulated as a QAP or one of its variants (Steinberg, 1961; Lawler, 1975; Pollatschek et al., 1976; Eiselt and Laporte, 1991; Bos, 1993; Phillips and Rosen, 1994; Miranda et al., 2005; Dell'Amico et al., 2009; Ünal and Uysal, 2014). Perhaps the most popular applications arise in logistics and facility layout (Dickey and Hopkins, 1972; Elshafei, 1977; Krarup and Pruzan, 1978; Fu and Kaku, 1997; Benjaafar, 2002; Chiang et al., 2002; Cordeau et al., 2006; Yener and Yazgan, 2019). These problems can also be adapted to formulate other fundamental problems such as the traveling salesman problem, the graph partitioning problem, the maximal clique problem, clustering and scheduling (Hansen and Lih, 1992; Pardalos et al., 1994; Malucelli, 1996).

An extensive review on formulations, solution techniques, and applications of QAPs is found in Abdel-Basset et al. (2018b). Exact methods usually fail to prove solution optimality within a reasonable time for instances of considerable size. Attempts to solve the QAP through a black-box solver using compact MILP formulations are found in Fischetti et al. (2012) and Zhang et al. (2013). An appropriate approach to deal with these instances is the use of metaheuristics. In a 2018 review, Abdel-Basset et al. (2018b) refer to 140 metaheuristics published to the QAP. This problem is customarily one of the first that new metaheuristic frameworks are adapted to, due to its simple formulation and difficulty of solving (Duman et al., 2012; Abdel-Basset et al., 2018a; Mihić et al., 2018; Dokeroglu et al., 2019). The most effective heuristics hybridize two or more metaheuristics, usually variants of the tabu search (TS), such as the robust TS (RoTS) (Taillard, 1991) and the concentric TS (CoTS) (Drezner, 2005b). The TS is the intensification component coupled with another metaheuristic, used as a diversification component, to generate new initial solutions for the search, such as random swaps (Misevičius, 2012), genetic algorithm (Drezner, 2003, 2005a), simulated annealing (Chiang and Chiang, 1998; Misevičius, 2003, 2004) and ant colony (Gambardella et al., 1999; Talbi et al., 2001). Many of these hybrid metaheuristics are population-based, such as memetic algorithms (Benlic and Hao, 2015; Harris et al., 2015), allowing multiple solutions to evolve in parallel. Therefore, parallel computing can be used to potentially linearly reduce the running time of the algorithm with the number of available cores.

The best performing algorithms found to solve large-size instances of the QAP apply some level of parallelism to hybrid metaheuristics, usually requiring little modifications to adapt the algorithm to a parallel computing environment. Parallelization in the QAP literature is used both to speedup exact methods, such as the branch-and-bound algorithms proposed in Pardalos and Crouse (1989) and Clausen and Perregaard (1997), and heuristics. The chunks of code usually parallelized in heuristics are the objective function evaluation – for example, when evaluating neighbor solutions during a tabu search (Van Luong et al., 2011) – or the application of one or several heuristics whenever evolving multiple solutions simultaneously. Due to the intense arithmetic computation level of the objective function evaluation, this process can be

greatly speeded up using Graphical Processing Units (GPUs), although parallelization using CPUs is also possible (Taillard, 1991). Czapiński (2013) reported speed ups of up to 300× in large size instances using the CUDA platform. However, several limitations exist to design an effective algorithm in a GPU environment related to memory management (Zhu et al., 2010). Nonetheless, some classical metaheuristics have been adapted to be run in GPUs (Tsutsui and Fujimoto, 2009; Poveda and Gómez, 2018; Sonuc et al., 2018). Another way to parallelize an algorithm is by using a cluster of machines with multiple nodes or a single CPU with multiple cores. In a shared-memory environment, it is common to use a cooperative heuristic, where a mechanism is used to exchange useful information between the parallel instances in the hope of accelerating the search process or improving the solution quality. Information is shared through a global memory that can contain, for example, the history of the search already performed by a tabu search (Talbi and Bachelet, 2006), the pheromones of ants in an ant colony (Talbi et al., 2001), all individuals to be crossed in evolutionary algorithms (Dokeroglu, 2015; Harris et al., 2015; Tosun, 2015; Munera et al., 2016), a reference set with the best individuals found so far to be used in the crossing over (James et al., 2005, 2009a), previously visited solutions to perform similarity checks avoiding searches in redundant areas of the solutions space (Aksan et al., 2017), the best solutions found in each node (Abdelkafi et al., 2019), or even the parameters of a selected metaheuristic in a hyper-heuristic framework (Dokeroglu and Cosar, 2016). In distributed-memory systems, only the development of independent algorithms is possible, where no information is exchanged between parallel instances, as in Clausen and Perregaard (1997). Despite successful applications of parallel hybrid algorithms for the QAP, no study was found to solve its variants using parallel computing.

The main challenge when designing a hybrid metaheuristic for the QAP is on the elaboration of appropriate diversification procedures for a more efficient search (Misevičius, 2019). In this paper, we present a parallel memetic iterated tabu search (PMITS) metaheuristic to solve the QAP, QBAP, BiQAP, QSAP and GQAP. The PMITS extends the iterated tabu search presented in Misevičius (2012), which is one of the best performing sequential (non-parallel) algorithms to solve the hardest benchmark instances found in the QAPLIB (Misevičius, 2019), to a parallel environment by adding: (i) a modified version of the randomized uniform-like crossover (RULX) operator (Misevičius and Kilda, 2005), called skewed crossover (SX), and an adaptive mutation operator to generate new solutions to be improved by the neighborhood searches; (ii) a two-phase neighborhood search, where in the first phase a 2-opt local search – for QAP, QBAP and BiQAP – followed by a drop/add local search – for QSAP and GQAP – are performed, so that the new solution is moved to a local optimum; (iii) an acceptance criterion based on the previous local optimum visited; (iv) a modified version of Taillard's RoTS to be used in the second phase of the neighborhood search, which saves all visited solutions in the tabu list, reducing the number of parameters to be set and the management of data in the tabu list; (v) a long-term memory where solutions previously found by the tabu search are cached to fast retrieval, speeding up the algorithm; and (vi) an anti-stagnation operator

that restarts the whole population when no new improving solution is found. We use parallel computing to perform the memetic operators (crossover and mutation), the local searches and the tabu search in multiple processes simultaneously. Cooperation between the parallel nodes occur when executing SX and verifying the acceptance criterion. We show that our framework is robust enough to be adapted to solve all five problems effectively requiring no major modifications. Computational experiments attest the effectiveness of PMITS. Our method has the best average performance compared to the state-of-the-art parallel metaheuristics for the hardest instance sets of QAPLIB for the QAP, finding solutions on average at 0.03% from best known solutions (BKS). For the QBAP, BiQAP, QSAP, and GQAP, we find the BKS in all instances tested significantly outperforming the state-of-the-art methods used as comparisons, either heuristics or exact methods. As another contribution of this paper, we present and compare exact approaches for the QAP variants using three different formulations – a quadratic integer programming (QIP) model, and two linearization techniques.

The remainder of this paper is organized as follows. Section 4.2 gives a formal presentation of the five problems considered by presenting their quadratic mathematical formulations and two techniques to linearize them. Section 4.3 reviews the best performing metaheuristics, both sequential and parallel, to solve each problem found in literature. Section 4.4 details the PMITS. The results of computational experiments for each of the problems are presented in Section 4.5. Finally, Section 4.6 presents our concluding remarks.

4.2 Mathematical formulations and literature review

The quadratic formulations for the QAP, QBAP, BiQAP, QSAP, and GQAP are presented next followed by two linearization techniques for the quadratic terms.

4.2.1 Quadratic Assignment Problem

Let n be the number of facilities to be assigned to n sites, where $\mathcal{N} = \{1, \ldots, n\}$ represents the set of such facilities and sites. Also, let $p(i)$ represent the site where facility $i \in \mathcal{N}$ is installed. Two $n \times n$ matrices are given. $\mathbf{F} = (f_{ij})$ represents the flow between facilities i and j, and $\mathbf{D} = (d_{kl})$ represents the distance between sites k and l. A simple way to describe the QAP is by a quadratic 0–1 formulation using a permutation matrix $\mathbf{X} = (x_{ij})$, such that:

$$x_{ik} = \begin{cases} 1, & \text{if } p(i) = k, \\ 0, & \text{otherwise.} \end{cases} \tag{4.1}$$

The matrix $\mathbf{X}$ is characterized by the following assignment constraints:

$$\sum_{i \in \mathcal{N}} x_{ik} = 1, \quad \forall k \in \mathcal{N}, \tag{4.2}$$

$$\sum_{k \in \mathcal{N}} x_{ik} = 1, \quad \forall i \in \mathcal{N}, \tag{4.3}$$

$$x_{ik} \in \{0, 1\}, \quad \forall i, k \in \mathcal{N}. \tag{4.4}$$

Hence, the QAP is formulated as:

$$\min \sum_{i \in \mathcal{N}} \sum_{j \in \mathcal{N}} \sum_{k \in \mathcal{N}} \sum_{l \in \mathcal{N}} f_{ij} d_{kl} x_{ik} x_{jl}, \tag{4.5}$$

subject to constraints (4.2)–(4.4), where each single product of f and d is the cost of assigning facilities i and j to sites k and l, respectively.

The QAP is a widely known combinatorial problem with a vast literature. We refer the reader to the reviews of Finke et al. (1987), Loiola et al. (2007), Burkard (2013) and Abdel-Basset et al. (2018b). Although the literature on exact approaches for the QAP is vast, benchmark instances with as few as 30 nodes are yet open to be solved (Burkard et al., 1997).

4.2.2 Quadratic Bottleneck Assignment Problem

In the bottleneck version of the QAP, the same inputs for the original problem are given. The objective, however, is to find a permutation $p \in \Pi_{\mathcal{N}}$ such that:

$$\min_{p \in \Pi_{\mathcal{N}}} \max_{i,j \in \mathcal{N}} f_{ij} d_{p(i)p(j)}, \tag{4.6}$$

which can be reformulated with a linear objective function by introducing a slack variable Z as:

$$\min Z, \tag{4.7}$$

subject to (4.2)–(4.4), and to

$$Z - f_{ij} d_{kl} x_{ik} x_{jl} \geq 0, \quad \forall i, j, k, l \in \mathcal{N}. \tag{4.8}$$

The literature of the QBAP is very limited. Burkard (2002) states that the QBAP is NP-hard since it contains the bottleneck travelling salesman problem as a special case. He also proves the asymptotic behavior of the problem. Although the QBAP can be solved using the same QAP algorithms, we could not find any study with computational results for the QBAP. Steinberg (1961) introduces QBAP for a wiring problem, but unfortunately the problem is not solved in the paper, only presenting the results for the given backboard wiring instance as a QAP.

4.2.3 Biquadratic Assignment Problem

In the BiQAP, we are given two four-dimensional arrays as input, $A = (a_{ijkl})$ and $B = (b_{p(i)p(j)p(k)p(l)})$, where a_{ijkl} is the QAP flow-equivalent parameter for the facilities i, j, k and

l, and $b_{p(i)p(j)p(k)p(l)}$ is the distance-equivalent parameter for the sites $p(i)$, $p(j)$, $p(k)$ and $p(l)$. The BiQAP is formulated in the Koopmans-Beckmann form (Burkard et al., 1994) as:

$$\min_{p\in\Pi_{\mathcal{N}}} \sum_{i\in\mathcal{N}}\sum_{j\in\mathcal{N}}\sum_{k\in\mathcal{N}}\sum_{l\in\mathcal{N}} a_{ijkl}b_{p(i)p(j)p(k)p(l)}. \tag{4.9}$$

N-adic variants of the QAP, such as the cubic and quartic versions – as the BiQAP is also called – are generalizations of the QAP, therefore, they are also NP-hard. Although the BiQAP has been considered since the 1960s, it was formally studied for the first time several decades later in Burkard et al. (1994), who present different formulations for the problem and show how to compute lower bounds for the optimal solution value, derived from lower bounds for the QAP. The asymptotic behavior of the BiQAP is proved by showing that the ratio between the worst and optimal solution values tends to one as the size of the problem increases.

4.2.4 Quadratic Semi-Assignment Problem

In the QSAP, we are given a possible distinct number of facilities and sites, where $\mathcal{M} = \{1,\ldots,m\}$ is the set of facilities and $\mathcal{N} = \{1,\ldots,n\}$ is the set of sites. Semi-assignment constraints state that only facilities should be assigned to sites, but sites can stay unassigned. Also, there is no constraint on the number of facilities assigned to each site. A location cost matrix $\mathbf{C} = (c_{ik})$, where each value represents the cost of assigning facility i to site k, is usually considered. The quadratic 0-1 formulation of the QSAP is given as:

$$\min \sum_{i\in\mathcal{M}}\sum_{k\in\mathcal{N}} c_{ik}x_{ik} + \sum_{i\in\mathcal{M}}\sum_{j\in\mathcal{M}}\sum_{k\in\mathcal{N}}\sum_{l\in\mathcal{N}} f_{ij}d_{kl}x_{ik}x_{jl}, \tag{4.10}$$

subject to

$$\sum_{k\in\mathcal{N}} x_{ik} = 1, \quad i\in\mathcal{M}, \tag{4.11}$$

$$x_{ik}\in\{0,1\}, \quad i\in\mathcal{M}, k\in\mathcal{N}. \tag{4.12}$$

The QSAP NP-hardness is proven in Sahni and Gonzalez (1976). Malucelli and Pretolani (1995), Malucelli (1996), Drwal (2014) and Panyukov and Shangin (2016) provide lower bounds for the QSAP and present certain polynomial solvable cases. Saito et al. (2009) study the polytope that arises from a MILP reformulation of the problem. Milis and Magirou (1995) propose a Lagrangean relaxation algorithm for this problem and show that even for a QSAP of small size, it is hard to find optimal solutions.

4.2.5 Generalized Quadratic Assignment Problem

The GQAP is the capacitated version of the QSAP. The arrays r_i, indicating the size of a facility i, and C_j, indicating the capacity of a site j, are given. The formulation of the GQAP,

as presented in Lee and Ma (2004), considers the objective function (4.10) subject to (4.11), (4.12) and to capacity constraints given as:

$$\sum_{i \in \mathcal{M}} r_i x_{ik} \leq C_k, \quad k \in \mathcal{N}. \tag{4.13}$$

Once again, only small instances could be solved to optimality in the GQAP. Lee and Ma (2004) present three linearization approaches and a branch-and-bound algorithm. Hahn et al. (2008a) describe a reformulation with a dual ascent procedure in a branch-and-bound scheme to prove optimality for instances adapted from another problem. Pessoa et al. (2010) find new lower bounds for the same instances using Lagrangean relaxations of the reformulation from Hahn et al. (2008a).

4.2.6 Linearization techniques

The quadratic terms in the presented models can be linearized using linearization techniques so that the problems can be solved by means of general purpose mixed integer linear programming solvers. Two common techniques are the standard linearization (Glover and Woolsey, 1974) and the reformulation-linearization technique (RLT) (Sherali and Adams, 2013).

To convert the QIP model presented in Section 4.2.1 for the QAP using the standard linearization technique, we first define linearization variables $z_{ikjl} = x_{ik}x_{jl}$, i.e., z_{ijkl} is equal to one if facility i is assigned to site k and facility j is assigned to site l, and zero otherwise. Then, the objective function (4.5) is replaced by:

$$\min \sum_{i \in \mathcal{N}} \sum_{j \in \mathcal{N}} \sum_{k \in \mathcal{N}} \sum_{l \in \mathcal{N}} f_{ij} d_{kl} z_{ikjl}, \tag{4.14}$$

and the following linear constraints are added to the model:

$$z_{ikjl} \leq x_{ik}, \quad \forall i, j, k, l \in \mathcal{N}, \tag{4.15}$$

$$z_{ikjl} \leq x_{jl}, \quad \forall i, j, k, l \in \mathcal{N}, \tag{4.16}$$

$$z_{ikjl} \geq x_{ik} + x_{jl} - 1, \quad \forall i, j, k, l \in \mathcal{N}. \tag{4.17}$$

$$0 \leq z_{ikjl} \leq 1, \quad \forall i, j, k, l \in \mathcal{N}. \tag{4.18}$$

While the advantage of the standard linearization technique lays on its simplicity, RLTs usually require the addition of fewer constraints to the model. The RLT consists of a reformulation step in which nonlinear valid inequalities are generated, and a linearization step in which each product term in the objective and constraints is replaced by a single variable. Considering again the linearization variables $z_{ikjl} = x_{ik}x_{jl}$, the level-1 RLT (RLT-1) formulation of QAP provided by Frieze and Yadegar (1983) considers the objective function (4.14) and adds the following constraints to the original model:

$$\sum_{i \in \mathcal{N}} z_{ikjl} = x_{jl}, \quad \forall j, k, l \in \mathcal{N}, \tag{4.19}$$

$$\sum_{j \in \mathcal{N}} z_{ikjl} = x_{ik}, \quad \forall i, k, l \in \mathcal{N}, \tag{4.20}$$

$$\sum_{k \in \mathcal{N}} z_{ikjl} = x_{jl}, \quad \forall i, j, l \in \mathcal{N}, \tag{4.21}$$

$$\sum_{l \in \mathcal{N}} z_{ikjl} = x_{ik}, \quad \forall i, j, k \in \mathcal{N}, \tag{4.22}$$

$$0 \leq z_{ikjl} \leq 1, \quad \forall i, j, k, l \in \mathcal{N}. \tag{4.23}$$

Both models (standard and RLT-1) can be reduced by considering common assumptions found in an instance structure. By assuming $f_{ii} = 0$ and $d_{kk} = 0$, we eliminate all constraints where $i = j$ and $k = l$. For a symmetric instance, i.e., $f_{ij} = f_{ji}$ and $d_{kl} = d_{lk}$, we can eliminate all variables z_{ikjl} with $i > j$ by making the substitutions necessary in the objective function and constraints. RLTs of higher levels can also be found and provide good overall results (Adams et al., 2007; Hahn et al., 2012). Although one can find other formulations and linearization techniques for the QAP in the literature (Loiola et al., 2007; Nyberg and Westerlund, 2012; Burkard, 2013; Abdel-Basset et al., 2018b), they are usually designed to perform well under specific conditions such as for special matrix structures (Çela et al., 2018). The advantage of the presented RLT-1 is to provide an easy to understand model that has a reasonable performance for any instance structure and an easy adaptability for the QAP variants. RLTs are found in the literature for some QAP variants. For the GQAP, an RLT-1 has been applied by Hahn et al. (2008a), where they show that this technique outperforms three other reformulations proposed by Lee and Ma (2004). For the QSAP, Billionnet and Elloumi (2001) and Schüle et al. (2009) present RLT-1 models similar to those described here, and Schüle et al. (2009) also provide higher levels RLT formulations.

4.3 State-of-the-art metaheuristics for the QAP and variants

In this section, we review the best performing metaheuristics, both sequential and parallel, found to solve the QAP and its variants.

4.3.1 Sequential metaheuristics for the QAP

Undoubtedly, the most well-studied algorithm to solve the QAP is the robust tabu search (RoTS) introduced by Taillard (1991). RoTS is an adaptation of tabu search to the QAP, which uses a special data structure as tabu list that forbids previously visited moves from a 2-opt local search. Most of the best-performing algorithms found to solve the QAP use, or adapt, RoTS as the intensification procedure. Most of its success is due to the fast evaluation of pairwise exchange neighbor solutions in QAPs (see more in Section 4.4.3) together with the easiness to be hybridized with many metaheuristics. Hybridization in QAP literature is commonly done in an iterated local search (ILS) framework, where RoTS, or any kind of local search variant, is used for intensification and another metaheuristic is used for diversification

(Stützle, 2006). The ITS presented by Misevičius (2012) succesfully improves RoTS by using a diversification operator that modifies the previous best solution found by a TS using a mutation operator mechanism based on multiple random swaps of facility assignments. While Misevičius' algorithm works with the evolution of a single solution, Benlic and Hao (2015) propose a memetic algorithm, called Breakout Memetic Algorithm (BMA), where a population of solutions evolve by applying crossover and mutation operators, followed by a special local search procedure, called Breakout Local Search (BLS), which is a special version of a TS adapted to the QAP, where the perturbation move is adaptively modified between random and guided (using a tabu list) moves. The crossover operator used, called uniform crossover, selects two solutions from the current pool and creates a new one with nearly 50% of genes from each parent. A modified version of this operator, developed to improve performance in our algorithm, is used in this work (see Section 4.4.3). The Teaching-Learning-Based Optimization metaheuristic is hybridized with RoTS (TLBO-RTS) in Dokeroglu (2015). This is a two-phase evolutionary algorithm where solutions are recombined first with a higher-quality solution and then with the other current solutions before being improved by RoTS. Other diversification procedures to generate solutions to be improved by RoTS and some variants are analyzed in James et al. (2009b) for a population-based algorithm.

4.3.2 Parallel metaheuristics for the QAP

Many of the best performing parallel metaheuristics for the QAP are based on the sequential algorithms. When Taillard (1991) introduced RoTS, he proposed to distribute the neighborhood search on many processors, dividing a neighborhood into same-sized parts, and evaluating each of these parts on different processors. He also presented a theoretical parallelization of multiple independent TS. The TLBO-RTS of Dokeroglu (2015) is also presented with a parallel version where different populations evolve independently on multiple island processors. The Parallel Hybrid Algorithm (PHA) of Tosun (2015) is performed in three stages. First, a population evolves in parallel islands using a genetic algorithm (GA) with occasional migrations of individuals between the islands. Then, a TS is used to diversify the solutions. Finally, solutions are improved using RoTS. James et al. (2009a) introduced the Cooperative Parallel Tabu Search (CPTS) which is a multi-start TS that applies a modified RoTS with different parameters for each processor. Initial solutions for the search are generated by using a global reference set which uses information exchange to promote intensification and diversification. The parallel BLS (BLS-OpenMP) by Aksan et al. (2017) evolves multiple solutions in parallel using the BLS of Benlic and Hao (2015). They implemented an operator that constantly checks the similarity of solutions to avoid redundancies on the starting points of the search. An Extremal Optimization (EO) procedure, an evolutionary algorithm which progressively eliminates the least fit solutions, is hybridized with RoTS and run in parallel (ParEOTS) by Munera et al. (2016). Improvements were achived when RoTS, EO and a GA run in parallel using a framework designed for cooperative parallel local searches (CPLS-GA) in López et al.

(2018a). While the CPLS framework is designed to handle single-solution metaheuristics, another framework was proposed to ease the implementation of hybrid metaheuristics using cooperative parallelism, called Parallel Hybridization of Simple Heuristics (PHYSH) in López et al. (2018b). Another metaheuristic hybridized with RoTS is the Artificial Bee Colony by Dokeroglu et al. (2019). In its parallel version (PABC-QAP), multiple bee hives generate different solutions in separate processors using short-run TSs. The best solutions generated in the hives are then improved by a long-run TS. While in all described methods the metaheuristics used to evolve solutions are static, Dokeroglu and Cosar (2016) present a MultiStart Hyper-heuristic (MSH-QAP) where four different methods – RoTS, BLS, simulated annealing and ant colony – can be used. An evolutionary mechanism is used in the primary node to evaluate the four heuristics and find near-optimal parameters for a given problem instance. In a second phase, the best performing heuristic is selected to perform several multistarts in parallel in order to improve the solution quality.

A good speedup can also be achieved when implementing RoTS in GPUs. Unlike CPUs, a GPU in the CUDA environment follows a *single instruction – multiple data* (SIMD) pattern, meaning that it executes the same instruction set on different data elements at the same time. Several threads run concurrently, each one executing the same program (kernel). Kernels can communicate with each other via a shared memory. Memory management should be done as effectively as possible due to current limitations. Zhu et al. (2010) present a parallel SIMD tabu search (SIMD-TS), using the concentric tabu search (CoTS) presented by Drezner (2003), designed for the unique environment found in GPUs. Independent TSs are assigned to each thread. By assigning each search to a thread, the algorithm can perform a full search without communicating with the CPU, which is a time expensive task. A random diversification operator is called after some TS iterations to avoid stagnation. An improved version of this algorithm is presented by Czapiński (2013). The Parallel Multistart Tabu Search (PMTS) implements the CoTS with two levels of parallelism, first for the neighborhood evaluation, as in Zhu et al. (2010), and second for the tabu list management and move selection, achieving a much higher speedup. An adaptation using RoTS with roughly equal performance is presented by Novoa et al. (2015). Although other QAP algorithms on GPU can be found (see a survey in Abdelkafi et al. (2016)), to the best of our knowledge these are the best performing ones.

All methods described in this and the previous subsections are tested on several instances from the QAPLIB library (Burkard et al., 1997), having very similar performances, and can be considered as being the best methods found to solve the QAP.

4.3.3 Metaheuristics for the QAP variants

BiQAP. Due to the difficulty to solve the BiQAP exactly, some heuristics have been proposed. Burkard and Çela (1995) develop and compare variants of simulated annealing, tabu search and other improvement methods, particularly using pairwise exchange algorithms. Computational

experiments using benchmark instances from Burkard et al. (1994) suggest that one version of the simulated annealing is the best among the variants tested. Later, Mavridou et al. (1998) describe a GRASP that iteratively constructs greedy solutions by making the first four assignments simultaneously using their cost of interaction, then assigns the other facilities one by one, and uses the pairwise exchange algorithm to improve the solution. This method was able to find the optimal solution for all the instances used in Burkard and Çela (1995) improving the performance of the simulated annealing, however, requiring large CPU times.

QSAP. The QSAP has many applications in scheduling, clustering, and partitioning problems (Domschke, 1989; Magirou and Milis, 1989; Hansen and Lih, 1992; Skutella, 2001). Some task-assignment problems can be formulated as QSAPs (Billionnet et al., 1992). Also, the hub location (Saito et al., 2009) and the metric labeling (Kleinberg and Tardos, 2002) problems are special cases of the QSAP. Any heuristic for these particular problems can be adapted to solve the QSAP. For the problem itself, Domschke et al. (1992) investigate three different management ideas to deal with the tabu list in a simple tabu search and compare them with simulated annealing.

GQAP. Heuristic approaches are also presented for the GQAP. Cordeau et al. (2006) present a memetic heuristic that combines genetic algorithms and tabu search, which solves Lee's instances faster and introducing larger instances. Mateus et al. (2011) propose several GRASP with path-relinking heuristics using different construction, local search, and path-relinking procedures. Later, McKendall and Li (2017) propose a simple tabu search heuristic to solve Cordeau's instances, and compare its performance with the other heuristics.

QBAP. Although any heuristic for the QAP potentially works for the QBAP, no study was found to explicitly consider it.

For none of these problems we could find implementations of the robust tabu search or any other of the main frameworks (parallelism, memetic operators) used in our PMITS.

4.4 Parallel memetic iterated tabu search

We now present our parallel memetic iterated tabu search (PMITS) metaheuristic. Following the successful diversification-intensification approach for QAPs, we integrate a diversification phase where multiple solutions evolve in parallel using crossover and mutation operators, then are improved by local searches, and, if the local optimum is different than the previous one, they are improved in an intensification phase using a tabu search adapted from the QAP for its variants.

4.4.1 Solution representation

A solution x for any of the five problems considered can be encoded as a vector $x = \{p(i)|i \in \mathcal{N}\}$, where $p(i)$ represents the site where facility i is installed, as previously defined. The five QAP variants considered here are split into two sets. The permutation problems are those in which x is represented by a permutation of the sites indices. The QAP, QBAP and BiQAP belong to this set. Meanwhile, the non-permutation problems are those that a site index can appear multiple times in x. This set contains the QSAP and GQAP.

4.4.2 Constructive heuristics

Three different heuristics are implemented to create initial solutions. On the permutation problems, we simply generate a random permutation of the sites, assigning each one to a facility. On the QSAP, every facility is assigned to a random site, until all facilities are located. To generate a feasible solution for the GQAP, additional steps are required to respect capacity constraints. The steps performed are as follows:

1. Generate a queue of unassigned facilities in a random order;
2. Assign the first facility in the queue to a randomly drawn site. If there is enough capacity available, the facility stays assigned to the site and is removed from the queue. Otherwise, draw another site, different than the ones already selected. Repeat this until the facility is successfully assigned to a site or all sites are checked;
3. If the facility cannot fit in any site, remove a random facility already assigned from its site and place it in the end of the queue. Then, recheck if the current facility fits in this site. Repeat this step until the current facility can be assigned to a site;
4. Repeat steps 2 and 3 until the queue of unassigned facilities is empty.

Due to the randomness of these steps, all combinations of assignments are reachable using this procedure. Therefore, given enough time, this constructive heuristics is capable of finding any feasible solution, if one exists. In the permutation problems and the QSAP, a feasible solution is generated in time $O(n)$ for an instance with n facilities. However, for the GQAP heuristic, if the feasible solution space is too restricted, such as in the case when capacity constraints are tight, the process of removing assigned facilities to give space to unassigned ones can take extra time until a feasible solution is found. We present next local search heuristics to improve feasible solutions, which are embedded within the proposed PMITS metaheuristic.

4.4.3 Diversification via memetic operators

The PMITS proposed in this study consists of the parallel evolution of multiple processes using a memetic algorithm. Each process represents a solution evolved through generations.

In total, P processes evolve in parallel representing a population of feasible solutions. In memetic algorithms, solutions are modified every generation by a crossover operator that exchanges information between known solutions and by a mutation operator, and they are subsequently improved by a neighborhood search. In our metaheuristic, each solution from the previous population is replaced in the current one by a new solution generated from the crossover of itself with another randomly chosen solution from the previous population using the crossover and mutation operators described in Section 4.4.3. Then, the new solution is improved by the local searches described in Section 4.4.3. The resulting local optimum is accepted only if it is a different solution than the original one from the previous population.

Crossover and mutation

Several crossover operators for the QAP are compared in Misevičius and Kilda (2005). Their results show that higher performance is obtained with crossovers with lower degree of disruption. From preliminary experiments, we decided to use in our PMITS a modified version of the randomized uniform-like crossover (RULX) in which the degree of disruption is reduced when generating new solutions. In the RULX, two solutions (parents) are selected from the current population. Then, from a randomized starting position within the encoded vectors of the parents, the value is selected from one of the parents with probability $\phi = 0.5$ from parent 1 and $1 - \phi = 0.5$ from parent 2. The operator moves to the next one following a randomly chosen direction. For the permutation problems, if the chosen value was already placed in another previously visited position, we change the value to the one of the other parent. If both were already assigned, the position is left empty. When the whole vector is iterated, the unassigned facilities are randomly assigned to the empty sites. The RULX can potentially generate solutions that are up to 50% different from each parent. To reduce the level of disruption, we modified RULX so that ϕ is biased, i.e., $\phi > 0.5$. We call this modified version of RULX as *Skewed Crossover* (SX). The value of ϕ is a parameter of SX. In PMITS, this probability is set dynamically. It starts from a fixed value, set from preliminary experiments to $\phi = \min\{0.9, (n-3)/n\}$ – to guarantee that, on average, at least three positions of the offspring are different than parent 1 – and it is reduced by 0.01 every time a solution fails to be accepted by the criterion previously mentioned.

The mutation procedure used consists of performing k random moves in the neighborhoods defined for the local searches. For the permutation problems, this means to perform k 2-opt moves, where each move is performed for a pair of facilities chosen randomly. For the non-permutation problems, a random move can be either in the 2-opt neighborhood or in the drop/add neighborhood with equal probability (see a description of these neighborhoods in Section 4.4.3). In the latter, a random facility is chosen and moved to a new random site, always respecting the problem constraints to keep the new solution feasible whenever applicable. The value of k is also set dynamically, starting at $k = 1$ and increasing by 1 every

time a solution is not accepted if mutation is used. This procedure is similar to the kicks in a variable neighborhood search. A mutation procedure has a fixed probability M of being called after a crossover. The value of M was set experimentally as shown in Section 4.5.3.

Local searches

Local search heuristic is a technique used to improve a solution by searching in the neighborhood of its solution space. In the proposed PMITS, we implement two of the most common procedures used in local searches for the QAP and its variants. The first one is well suited for all problems studied here and is based on swapping the positions of two facilities. The second one is applicable only to the non-permutation problems, and consists of the movement of facilities between sites.

Pairwise exchange local search. Local search 1 (LS1) performs the *pairwise exchange* procedure (Fleurent and Ferland, 1994; Skorin-Kapov, 1990). If solution $x = \{p(i)|i \in \mathcal{N}\}$ is a starting point for LS1, the neighbor $x' = \{q(i)|i \in \mathcal{N}\}$ obtained by the swap of facilities of indices r and s is:

$$\begin{aligned} q(i) &= p(i), \quad \forall i \in \mathcal{N} \setminus \{r, s\}, \\ q(r) &= p(s), \\ q(s) &= p(r). \end{aligned}$$

The neighborhood of LS1 ($\mathbf{N}_1(x)$) contains the solutions generated by all swaps between r and s such that $r < s$. The objective of LS1 is to find the best (hopefully improving) solution x^* contained in the neighborhood of x, i.e., $x^* = \arg\min\{f(x')|x' \in N_1(x)\}$, also known as steepest descent. In our heuristic, an exhaustive search is performed, meaning that every time a solution is improved a new local search is performed starting from this new solution. The local search stops when no improving solution can be found in a neighborhood.

In the permutation problems, all swaps will result in a feasible and distinct solution, meaning that a neighborhood contains exactly $\frac{n(n-1)}{2}$ solutions. In the QSAP, when two different facilities are located in the same site, their swap does not change the current solution. So, LS1 neighborhood size in this problem is bounded by $\frac{n(n-1)}{2}$; the actual size depends on the number of distinct sites where facilities are located. For example, in the case that all facilities are assigned to the same site, then $\mathbf{N}_1(x) = \{\emptyset\}$. In the GQAP, the neighborhood size can be even smaller. Beyond distinct locations between swapped facilities, we should also consider if the swap results in a feasible solution. In a swap between any pair of facilities i and j with capacities $r_i \geq r_j$ the neighbor generated is feasible if $r_i - r_j \leq C_{p(j)} - \sum_{k|p(k)=p(j)} r_k$, i.e., if there is enough capacity left in the site to accommodate the extra space of the new facility it is receiving.

In each iteration of LS1, we are interested in calculating the difference $\Delta(x, r, s) = f(x') - f(x)$, where $x' \in \mathbf{N}_1(x)$ is obtained by swapping facilities r and s in the current solution x. The simplest way to calculate $\Delta(x, r, s)$ is by calculating separately $f(x)$ and $f(x')$, then performing the subtraction. For the QAP, this would take $O(n^2)$ because the calculation of $f(x)$ for any feasible solution x takes $O(n^2)$ time. However, when using the pairwise exchange local search for the QAP, one particular effect is that the cost of a neighbor solution can be computed in $O(n)$ operations, instead of $O(n^2)$. For symmetric matrices $\mathbf{F}$ and $\mathbf{D}$ with null diagonal, the computation is as follows (full formula can be found in Taillard (1991) and Benlic and Hao (2013)):

$$\Delta(x, r, s) = \sum_{i \in \mathcal{N} \setminus \{r,s\}} 2(f_{ri} - f_{si})(d_{p(r)p(i)} - d_{p(s)p(i)}). \tag{4.24}$$

Taillard (1991) proposed to use memory to store the values $\Delta(x, r, s)$ so that if x' is the solution obtained after swapping the sites of facilities r and s in the solution x, then computing $\Delta(x', u, v)$ is even faster if pairs $\{r, s\}$ and $\{u, v\}$ are mutually exclusive, i.e., $\{r, s\} \cup \{u, v\} = \{\emptyset\}$. Thus, $\Delta(x', u, v)$ is calculated in $O(1)$ as:

$$\Delta(x', u, v) = 2(f_{su} - f_{sv} + f_{rv} - f_{ru})(d_{q(s)q(u)} - d_{q(s)q(v)} + d_{q(r)q(v)} - d_{q(r)q(u)}) + \Delta(x, u, v). \tag{4.25}$$

Misevičius (2012) presented a trick to speed up these computations using three-dimensional matrices to retrieve the values of the subtractions performed that was also implemented in our algorithm.

The same reductions are applicable to the QSAP and GQAP with the addition of the differences in the sum of location costs. An adaptation for BiQAP also implemented is shown in Burkard and Çela (1995), where the objective function computation is reduced from $O(n^4)$ to $O(n^3)$ time by calculating the difference between a solution and its neighbor, and to $O(n^2)$ by storing the values $\Delta(x, r, s)$ to calculate neighbor objective functions after the first iteration. We refer the reader to Burkard and Çela (1995) to see the full detailed reduced equations for the BiQAP. The same principle can be used to adapt these formulas to reduce the evaluation time for QBAP to $O(n)$ after a swap between facilities r and s, where the new bottleneck is changed to facilities $k \in \mathcal{N}$ and r (or s) if $f_{kr}d_{kr}$ (or $f_{ks}d_{ks}$) is greater than the current bottleneck value.

Drop/add local search. Local search 2 (LS2) performs drop and add procedures to generate neighbor solutions for the non-permutation problems. This represents the exchange of the location of a facility. The neighborhood $\mathbf{N}_2(x)$ of a solution x explored by LS2 contains all solutions where facility $i \in \mathcal{N}$ is moved from its current site $p(i)$ to another site $k \in \mathcal{M} \setminus \{p(i)\}$.

In the QSAP, every facility movement will result in a feasible and distinct solution. Since each facility can be moved to $m - 1$ different sites, LS2 neighborhood of a QSAP solution contains

exactly $n(m-1)$ solutions. In the GQAP, it is necessary to verify capacity restrictions to ensure the procedure results in a feasible solution.

Now define $\Delta(x, r, p(r), p'(r)) = f(x') - f(x)$, where $x' \in \mathbf{N}_2(x)$ is obtained by moving facility r from its current site $p(r)$ to a new site $p'(r)$ in the current solution x. The evaluation of QSAP and GQAP solutions after a drop/add move can also be done in $O(n)$ in the symmetric case by computing

$$\Delta(x', r, p(r), p'(r)) = (c_{rp'(r)} - c_{rp(r)}) + \sum_{i \in \mathcal{N} \setminus \{r\}} 2f_{ri}(d_{p(r)p(i)} - d_{p'(r)p(i)}), \tag{4.26}$$

where the first subtraction refers to the differences in the location costs between the old and the new sites, while the second term of the equation refers to the differences in the assignment costs between r to each other facility.

4.4.4 Intensification via tabu search

The tabu search (TS) is undoubtedly the most popular and successful metaheuristic to solve the QAP. It consists in allowing solutions to move to the least degrading solution when no improving neighbor exists. In order to avoid cycling, the return to the previous solutions is forbidden. A data structure called *tabu list* is maintained indicating all forbidden moves. This list contains elements defining forbidden moves. In the QAP, a forbidden move can be defined by the pairs of facilities recently swapped (Skorin-Kapov, 1990) or by prohibiting moves where both facilities would be assigned to sites they had occupied in recent iterations, as in the succesful Taillard's RoTS (Taillard, 1991). We implemented a modified version of RoTS where we simply save all solutions previously visited during the current call of the search in the tabu list, thus eliminating parameters such as the tabu list size, the aspiration criteria and the number of iterations to keep a solution as tabu. Note that this might increase the complexity order to verify if a solution is in the tabu list, but it prevents the same solution being visited more than once during a single search. The only parameter to be set is the stopping criterion, chosen as the maximum number of iterations I performed by the TS without improvement in the best solution. The improving mechanism used is the pairwise exchange local search (LS1). Therefore, we can summarize the steps of our TS as:

1. Search for the best solution that is not in the tabu list using LS1 and change the current solution to it;
2. Add the new current solution to the tabu list;
3. If the current solution is the best one found so far, restart the iteration counter;
4. Repeat steps 1–3 until the iteration counter reaches I.

Since the TS procedure described is deterministic, we can save time by avoiding to explore a neighborhood already searched by caching initial solutions and their best solution found after TS. In our algorithm, we save solutions obtained in every run of TS in a map. Maps are data structures that associate a key to a value (Cplusplus, 2020). In the map created, the key is the initial solution and the value is its local optimum found by TS. Computationally, the search for an element in a map is quickly performed in $O(|\mathcal{N}| \log k)$, since they are typically implemented as balanced binary search trees, where k is the number of keys contained in the map and $|\mathcal{N}|$ refers to the size of the key. Thus, the process of retrieving a solution from the map is significantly faster than performing all steps from TS again. Although the same process could also be applied to individual local searches, the huge number of possible keys and the low repeatability of initial solutions after crossovers makes it computationally inneficient.

The pseudocode of PMITS is presented in Algorithm 4. For each of the P parallel evolutionary processes, we are interested in three solutions: the current solution (x), the previous local minimum (x'), and the best solution found so far (x^*). Each process is sent to a secondary node using a dynamic load balancing method. Whenever a secondary node finishes the execution of its process, it sends a message to the primary node to request a new process. The first step of PMITS consists in creating a feasible solution for each process using the constructive heuristics (line 5). Then, they are improved in parallel secondary nodes using TS (lines 6–8). Initially, all three solutions are equal, i.e., $x = x' = x^*$ (line 9). The perturbation phase comes next. First, we reset the number of random swaps for mutation (k) and the skewed probability (ϕ) for each thread. For each parallel secondary node, the solution x is modified by the SX operator (line 14) and by the mutation operator, if mutation is selected (lines 16–19). The new solution is improved using the local searches LS1 and LS2, respectively and whenever applicable, to reach a local minimum (line 20). If the current local minimum x is different than the previous local minimum x', then the solution is accepted (lines 21–28). If this solution is an existing key in the map, the new solution is retrieved from its associated value (lines 22–23). Otherwise, the TS is applied to improve x and the result is saved in the map (lines 25–26). If the solution is not accepted in line 21, the whole perturbation process for this thread p is repeated (lines 13–32). The best solution found in p is updated, if necessary (lines 29–31). An operator to avoid the stagnation behavior of the search after long trials restarts the whole population every thousandth generation after the last improvement in the best solution. In our PMITS, we consider up to three stopping criteria that depends on the resources available (line 10). The first is the maximum number of iterations, i.e., the number of times the perturbation process is repeated (lines 10–34). The second is the maximum running time. Finally, we observed in preliminary experiments that when multiple solutions converge from different start points to a similar local optimum, there is a very high chance that this solution is the global optimum. Thus, we defined a third stopping criteria stating that if at least 50% of the solutions in $\mathcal{X}^*$ are equal to the best solution found so far, then the search stops and the algorithm returns the best solution in $\mathcal{X}^*$. This last criterion is best suited to

use when P is high enough so that more solutions need to converge to avoid a false positive.

Algorithm 4 Parallel memetic iterative tabu search (PMITS)

Population size: P; Mutation rate: M; Number of TS iterations: I; {`Parameters`}
Set of best solutions: $\mathcal{X}^* = \{x_1^*, \ldots, x_P^*\}$;
Set of current local optimum solutions: $\mathcal{X}' = \{x_1', \ldots, x_P'\}$;
Set of modified solutions: $\mathcal{X} = \{x_1, \ldots, x_P\}$;
Create $\mathcal{X}^*$ using the constructive heuristics;
for all parallel $p \in P$ **do**
    $x_p^* \leftarrow TS(x_p^*, I)$; {`Perform TS on each solution`}
end for
$\mathcal{X}' \leftarrow \mathcal{X}^*$ and $\mathcal{X} \leftarrow \mathcal{X}^*$;
while stopping criteria are not satisfied **do**
    Set $k_p = 1$ and $\phi_p = \min\{0.9, (n-3)/n\}$ for each solution
    for all parallel $p \in \{1, \ldots, P\}$ **do**
        repeat
            $x_p \leftarrow SX(x_p', rand(\mathcal{X}'), \phi)$; {`Crossover current solution with a random one from the previous population`}
            $\phi_p \leftarrow \phi_p - 0.01$;
            if $rand(0,1) \leq M$ **then**
                $x_p \leftarrow Mutation(x_p, k_p)$; {`Mutate current solution`}
                $k_p \leftarrow k_p + 1$;
            end if
            $x_p \leftarrow LS(x_p)$; {`Search for local optimum`}
            if $x_p \neq x_p'$ **then**
                if $x_p \in map$ **then**
                    $x_p' \leftarrow$ Solution associated to key x_p in map; {`Retrieve TS solution from map`}
                else
                    $x_p' \leftarrow TS(x_p)$; {`Apply TS on the local optimum`}
                    Save key x_p with solution x_p' in map; {`Save TS solution in map`}
                end if
            end if
            if $f(x_p') < f(x_p^*)$ **then**
                $x_p^* \leftarrow x_p'$; {`Update best solution`}
            end if
        until $x_p \neq x_p'$
    end for
    if the last generation that the best solution in $\mathcal{X}^*$ was updated > 1000 **then**
        Recreate $\mathcal{X}'$ using the constructive heuristics;
    end if
end while
return best solution in $\mathcal{X}^*$.

4.5 Computational experiments

This section provides the results obtained from extensive computational experiments performed to setup our PMITS and to compare its performance with state-of-the-art methods to solve the QAP and its variants. We used computers equipped with an Intel Gold 6148 Skylake CPU with a 2.4 GHz clock, 8 GB of RAM and 40 cores. The PMITS was implemented in C++, compiled with the Intel compiler icpc, while the exact models were solved using CPLEX 12.5. The parallelism was implemented using OpenMP, which allows the implementation of shared-memory parallel programming as used in PMITS. The executable files of our codes can be

found at https://github.com/afcsilva/PMITS-for-QAPVar.git. All instances and results are available from https://www.leandro-coelho.com/quadratic-assignment-problems/.

4.5.1 Test instances

The QAP is solved using three sets of classical instances from the QAPLIB benchmark library. They are the *Taia* (unstructured, randomly generated), *Taib* (real-life-like instances) and *Sko* (instances with grid-based distances) sets. Each instance is labeled using the name of the set and the number of facilities contained in it. Together, the three sets consist of 31 problem instances with up to 100 facilities and sites. Although the library provides more instances, the others are easily solvable by many existing methods, thus they are not considered here. We did not find any instance set specifically designed for the QBAP. Thus, we used the same QAP sets for this problem, including six other smaller instances from Taia and Taib. For the BiQAP, the original instances used in Mavridou et al. (1998) are reported to be lost after contact with one of the authors. Thus, we generated 12 new instances using a random uniform distribution between 1 and 9 to determine each distance between sites and each flow between facilities considering both as symmetric matrices. The new instances have similar sizes as those from Mavridou et al. (1998), i.e., 10 to 36 sites and facilities. The GQAP is solved using a set of 21 instances taken from Cordeau et al. (2006), with up to 50 facilities and 20 sites. These instances are labeled with a code consisting of three parts: the number of facilities, the number of sites, and the tightness of the capacity constraints. Other smaller instances for this problem are found in Mateus et al. (2011), but they are easily solved using the existing methods, thus they also are not considered here. For the QSAP, we consider two sets of instances. Set 1 is an adaptation from the 21 instances of Cordeau et al. (2006) for the GQAP with no facility capacities. In these instances, distances between a facility to itself is zero. Since sites in the QSAP are uncapacitated, this would result in a solution with all facilities located in a single site, which gives a solution equal to zero. To prevent this, we set d_{jj} for $j \in \mathcal{M}$ to be equal to 50% of the average graph distance. Set 2 has 20 instances generated as follows. A number r of points in an Euclidean space of size 100×100 are generated where 70% of them represent facilities, i.e., $|\mathcal{N}| = 0.7r$, and the remaining represent sites, i.e., $|\mathcal{M}| = 0.3r$. Instances are generated for $r \in \{50, 75, 100, 125, 150\}$ points. The flow matrix $\mathbf{F}$ is generated uniformly from the interval $[q, 100]$, where $q \in \{1, 10, 25, 50\}$. The distance matrix $\mathbf{D}$ is the Euclidean distance matrix. Distances d_{jj} for $j \in \mathcal{M}$ are set in the same way as for the set 1. The assignment costs c_{ij} are set to $\delta_{ik} d_{ik}$, where δ_{ik} is randomly generated using a uniform distribution from $[q, 100]$ for $q \in \{1, 10, 25, 50\}$. The instances are identified by the number of points r and class q. While instances in set 1 are all symmetric, in the new set they are all asymmetric.

4.5.2 Testing the exact models for the QAP and variants

We adapted the standard reformulation technique and the RLT-1 presented for the QAP to all four variants considered in this paper. Our goal is to determine which formulation (original

QIP models, standard linearization, or RLT-1) performs better for each problem. Results for these experiments are shown in C.1. For the MILP solver used, we observed that, for the QAP, the QIP model works best for the instances tested. For the QBAP, RLT-1 presented the best performance. For the QSAP and the GQAP, the standard technique performed usually better for smaller instances while the QIP was better for larger instances. We highlight that our results show that the standard technique can potentially outperform the state-of-the-art RLT-1 formulation used in Pessoa et al. (2010) for the GQAP. From the six instances that standard solved that were used in their experiments, five were solved by their best RLT-1 model. The average time to solve them was 100 minutes for their RLT-1 model, but only two minutes by our standard linearization model. Also, the standard model was capable of solving an instance with 40 facilities, which is the largest GQAP instance with optimality proven. Finally, no instance of the BiQAP could be solved within the time limit using any formulation.

4.5.3 Parameter setting

Two rounds of experiments were performed to tune our PMITS. We first set the parameters P (population size), M (mutation rate), and I (number of TS iterations) using a selected number of instances. As reported by other studies, the performance of QAP algorithms is influenced by the characteristics of the solutions space landscape (Benlic and Hao, 2015; Dokeroglu, 2015). Thus, we individually set parameters for each QAP benchmark set using a pair of instances from each one chosen using the criteria of picking those that are neither too easy nor too hard to solve in a short run according to observations during preliminary experiments. These are: *tai40a* and *tai50a* for Taia, *tai80b* and *tai100b* for Taib, and *sko81* and *sko90* for Sko. The same criteria was used to chose three instances for each of the remaining problems. These are: *tai15a*, *tai20a*, and *tai25a* for the QBAP; $|\mathcal{N}| \in \{18, 20, 22\}$ for the BiQAP; *35-15-95*, *40-09-95*, and *50-10-95* for the GQAP; and *100-C1*, *125-C10*, and *150-C25* for the QSAP. Then, we analyze our metaheuristic speedup when using the parallelism by presenting a scalability analysis to determine the number of CPU cores to use in the search.

Setting the population size, mutation rate and number of tabu search iterations

We performed a grid search to determine which combination of the parameters P (population size), M (mutation rate) and I (number of TS iterations) leads to a faster convergence of the best solution found by our PMITS. We increased P from 40 to 1,000, M from 0 to 1, and I from 0 to 500, using four different values for each one, resulting in 64 combinations of parameters values. Tests were replicated 20 times for each instance with each of the combinations tested. Only the time limit was considered as a stopping criterion, being 10 seconds for the QSAP and GQAP, one minute for the QAP and QBAP, and five minutes for the BiQAP. The results are presented in Tables 4.1 and 4.2. The average percent deviations (APD) are computed as

the average relative distance between the average solution of the 20 replications and the best known solution (BKS) of each instance.

Table 4.1 – Parameter setting of PMITS: average percentage deviation for QAP instances

P	M	*Taia*				*Taib*				*Sko*			
		$I=0$ APD	$I=50$ APD	$I=200$ APD	$I=500$ APD	$I=0$ APD	$I=50$ APD	$I=200$ APD	$I=500$ APD	$I=0$ APD	$I=50$ APD	$I=200$ APD	$I=500$ APD
40	0.0	0.64	0.20	0.20	0.20	0.04	0.02	0.04	0.04	0.07	**0.04**	0.06	0.07
	0.1	0.65	0.19	0.18	0.18	0.04	0.03	0.03	0.03	0.08	**0.04**	0.05	0.07
	0.5	0.68	0.22	0.20	0.15	0.02	0.02	0.03	0.02	0.07	**0.04**	0.05	0.06
	1.0	0.73	0.22	0.17	0.19	0.02	0.01	0.02	0.02	0.06	**0.04**	0.05	0.06
100	0.0	0.62	0.16	0.19	0.17	0.01	0.01	0.01	0.03	0.06	**0.04**	0.06	0.07
	0.1	0.63	0.16	0.17	0.18	0.01	0.01	0.01	0.03	0.06	**0.04**	0.05	0.07
	0.5	0.65	0.18	0.18	0.17	0.01	0.01	0.01	0.02	0.07	**0.04**	0.05	0.07
	1.0	0.72	0.19	0.17	0.16	0.01	**0.00**	0.01	0.02	0.06	**0.04**	0.05	0.07
400	0.0	0.60	0.16	0.15	**0.12**	0.01	0.01	0.02	0.05	0.08	**0.04**	0.06	0.08
	0.1	0.57	0.16	0.16	0.16	0.01	0.01	0.02	0.05	0.08	**0.04**	0.06	0.09
	0.5	0.66	0.17	0.15	0.13	**0.00**	**0.00**	0.02	0.06	0.08	**0.04**	0.05	0.09
	1.0	0.64	0.16	0.13	0.13	**0.00**	**0.00**	0.02	0.06	0.07	**0.04**	0.06	0.08
1000	0.0	0.53	0.17	0.13	0.15	0.01	0.01	0.05	0.13	0.08	0.05	0.08	0.12
	0.1	0.59	0.17	0.17	0.16	0.01	0.01	0.05	0.12	0.07	0.05	0.07	0.12
	0.5	0.61	0.17	0.14	0.14	**0.00**	0.01	0.05	0.11	0.08	0.05	0.08	0.12
	1.0	0.66	0.17	0.16	0.15	**0.00**	0.01	0.04	0.10	0.07	**0.04**	0.07	0.12

Table 4.2 – Parameter setting of PMITS: average percentage deviation for QAP variants instances

P	M	*QBAP*				*BiQAP*				*QSAP*				*GQAP*			
		$I=0$ APD	$I=50$ APD	$I=200$ APD	$I=500$ APD	$I=0$ APD	$I=50$ APD	$I=200$ APD	$I=500$ APD	$I=0$ APD	$I=50$ APD	$I=200$ APD	$I=500$ APD	$I=0$ APD	$I=50$ APD	$I=200$ APD	$I=500$ APD
40	0.0	2.06	1.99	2.44	2.78	0.19	0.09	0.09	0.09	6.61	6.32	6.56	6.83	0.94	0.14	0.26	0.37
	0.1	2.15	2.12	2.29	2.95	0.18	0.09	0.09	0.09	0.64	0.10	0.06	0.08	0.47	0.06	0.19	0.24
	0.5	2.07	2.14	2.53	2.79	0.17	0.10	0.09	0.09	0.04	0.02	0.02	0.03	0.34	0.04	0.14	0.16
	1.0	2.05	2.27	2.63	2.88	0.17	0.08	0.09	0.09	0.03	0.01	0.03	0.04	0.24	0.01	0.08	0.06
100	0.0	1.92	2.07	2.44	2.91	0.16	0.08	0.08	0.08	5.06	5.60	6.11	6.35	0.70	0.10	0.23	0.32
	0.1	2.03	2.05	2.34	2.91	0.17	0.08	0.08	0.08	0.08	0.04	0.02	0.07	0.35	0.07	0.17	0.14
	0.5	1.95	2.06	2.62	2.94	0.16	0.09	0.08	0.08	0.03	**0.00**	**0.00**	0.01	0.26	**0.00**	0.05	0.10
	1.0	2.11	1.98	2.66	2.95	0.14	0.08	**0.07**	0.08	0.03	0.01	0.02	0.01	0.21	0.01	0.03	**0.00**
400	0.0	1.60	1.80	2.23	2.96	0.14	0.08	0.08	0.08	2.23	3.45	4.15	3.79	0.45	0.03	0.17	0.16
	0.1	1.65	2.00	2.32	2.71	0.16	**0.07**	**0.07**	**0.07**	0.09	0.02	0.07	1.37	0.31	0.02	0.12	0.16
	0.5	1.68	1.83	2.53	2.86	0.14	**0.07**	**0.07**	**0.07**	0.03	0.01	0.01	1.38	0.24	0.01	0.03	0.11
	1.0	1.63	2.00	2.46	2.79	0.14	**0.07**	0.08	**0.07**	0.03	0.01	0.01	0.87	0.22	**0.00**	**0.00**	0.03
1000	0.0	1.59	1.86	2.30	2.72	0.15	**0.07**	**0.07**	**0.07**	0.84	1.31	2.01	1.53	0.42	0.01	0.14	0.22
	0.1	1.63	1.88	2.25	2.53	0.15	**0.07**	**0.07**	**0.07**	0.09	0.04	1.03	1.34	0.29	**0.00**	0.11	0.17
	0.5	**1.58**	1.87	2.59	2.74	0.14	**0.07**	**0.07**	**0.07**	0.05	0.02	0.69	1.73	0.25	**0.00**	0.01	0.14
	1.0	1.76	1.89	2.41	2.95	0.14	**0.07**	**0.07**	**0.07**	0.04	0.02	0.19	0.78	0.17	**0.00**	0.01	0.05

The results show how some of the components of the algorithm behave when solving each problem. For the QAP, Table 4.1 shows that runs using a longer TS are more desirable for Taia, while for Sko shorter TS runs and, consequently, more crossing overs attain better results. For Taib, it is even possible to achieve good results with TS turned off. For the QBAP, we see from Table 4.2 that TS should be turned off and the population size should be large for better results. For the BiQAP, it is important to have TS on, regardless of the number of iterations performed among those evaluated, with a high population. For the QSAP, a small population with average number of TS iterations is the best option. For the GQAP, several different combinations attained good results, with highlight to those with a medium to large population and average number of TS iterations. Overall, mutation rates only affect

results significantly on QSAP and GQAP, where it is clearly better to have more mutations happening.

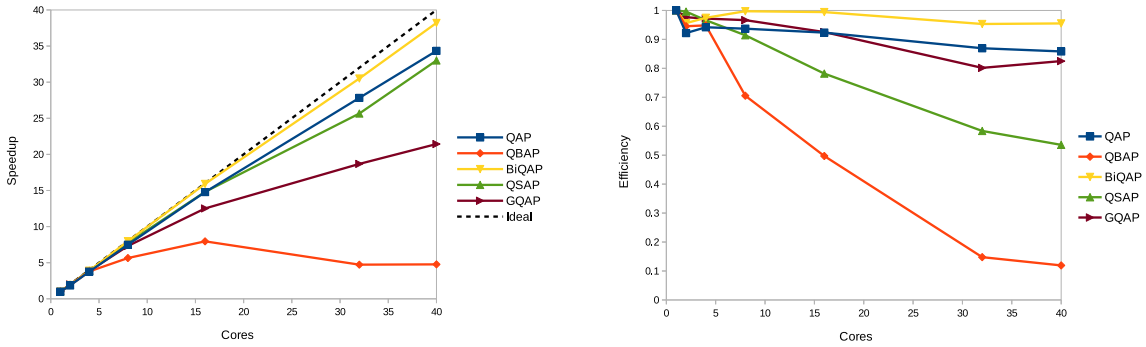
For the remaining experiments, we chose the following parameter values based on these results:

- QAP: $P = 400$, $M = 0$ and $I = 500$ for Taia, and $P = 100$, $M = 1$ and $I = 50$ for Taib and Sko;
- QBAP: $P = 1000$, $M = 0.5$ and $I = 0$;
- BiQAP: $P = 1000$, $M = 0.5$, and $I = 200$;
- QSAP: $P = 100$, $M = 0.5$ and $I = 50$;
- GQAP: $P = 100$, $M = 0.5$ and $I = 50$.

Setting the number of cores

We run a scalability analysis to verify the speedup achieved with parallelization in our code for each problem using the parameters as previously set and the instances used in the parameter setting. We use a fixed number of initial solutions (population size) and compare the computing time (wall-clock seconds) needed by PMITS to finish the exploration of their neighborhoods using a maximum number of cycles as the sole stopping criterion represented by the number of iterations performed by PMITS (lines 10–37 in Algorithm 4). The number of cycles was set so that run time using 40 threads are approximately those used in the parameter setting. We run for $T \in \{1, 2, 4, 8, 16, 32, 40\}$ cores. Figure 4.1a presents the speedup obtained in each problem, while Figure 4.1b shows the efficiency of the parallelism. The speedup obtained when running with T cores is calculated as $S_T = r_1/r_T$, where r_T is the wall-clock time for the algorithm with T cores. Ideally, a parallel algorithm would speed up the search the same number of times as the number of parallel cores used. In practice, S_T is upper bounded by T due to the time lost with the parallel overheads and the tasks performed outside of the parallelized code. A measure of efficiency in the scalability analysis is thus calculated as $E_T = S_T/T$, where E_T is the percentage of speedup obtained by the run with T cores compared to the ideal case.

The results demonstrate that in all problems, even considering their different parameter settings and instance sizes, paralellism can significantly speedup the run. Except for the QBAP, speedup increases when the number of cores is up to 40. This indicates that the performance could be further improved by using a larger number of cores if available. Speedup efficiency in the BiQAP, QAP and GQAP are above 80% with 40 cores. Efficiency drops quite steady down to 54% with 40 threads for the QSAP and to only 13% for the QBAP, likely because TS is turned off. It is relevant to notice that none of the best performing parameter settings have P equal to the number of CPU cores of our computational environment. The main reason



(a) Speedup

(b) Efficiency

Figure 4.1 – Scalability analysis for the parallelism in the PMITS

for this is that the run time of TS, which is the most time-consuming process of PMITS, is variable, which reduces the efficiency of core usage in a single process per core setting. For the next experiments, we used 40 cores for the QAP, BiQAP, QSAP and GQAP, and 16 cores for the QBAP, which are the number of cores that achieve the peak in speedups observed.

4.5.4 Comparison of PMITS against state-of-the-art methods

We present next the performance of our PMITS for all instance sets described in Section 4.5.1. In all but the GQAP, as stopping criteria, we used a time limit of one hour or a convergence criteria of 50% of the parallel solutions being equal to the best solution found so far. The maximum number of cycles is unlimited. PMITS is executed 20 times for each instance. All state-of-the-art methods used as comparison are those introduced in Sections 4.2 and 4.3.

Comparison for the QAP

A version of PMITS without paralellism, or simply MITS, is compared to the sequential algorithms JRG-DivTS (James et al., 2009b), ITS (Misevičius, 2012), TLBO-RTS (sequential) (Dokeroglu, 2015), and BMA (Benlic and Hao, 2015) in Table 4.3. With parallelism on, PMITS is compared to CPTS (James et al., 2009a), TLBO-RTS (parallel) (Dokeroglu, 2015), PHA (Tosun, 2015), MSH-QAP (Dokeroglu and Cosar, 2016), BLS-OpenMP (Aksan et al., 2017), PHYSH-QAP (López et al., 2018b), and PABC-QAP (Dokeroglu et al., 2019) in Table 4.4. Results are reported as the APD to the BKS, calculated as the average of $APD(x) = \frac{f(x) - f(x^*)}{f(x)}$ where $f(x)$ is the solution obtained by the heuristic and $f(x^*)$ is the BKS. Wall-clock time in minutes are reported only for reference since the algorithms were run on different computation environments. Overall, for the sequential algorithms, BMA presented the best performance followed by our MITS, which considerably improves ITS that inspired our metaheuristic. For the parallel algorithms, since most rely on different versions of a TS, as intensification procedure, we observed similar performances. For average size *Taia* instances

(*tai40a*, *tai50a* and *tai60a*), PHA, BLS-OpenMP and PMITS have the best performances. For the larger ones (*tai80a* and *tai100a*), PMITS and PHYSH-QAP are the best ones. *Taib* instances have a 0.00% gap in all runs with most algorithms, as well as for *Sko* instances. Analyzing both tables together, note that PMITS improves the average APD from 0.06% to 0.03% compared to MITS, reducing the number of instances without a 0.00% APD from 14 to six. Also, it is interesting to observe that although BLS outperforms MITS, its parallel adaptation (BLS-OpenMP) is outperformed by PMITS. A possible explanation lies on the chosen parameters for our algorithm, which used a large population size to benefit from speedup provided by paralellism. One would expect that for a sequential search, a better population size would be a smaller one. GPU implementations, such as SIMD-TS (Zhu et al., 2010) and TS (Novoa et al., 2015), have a considerably worse performance than the algorithms in Table 4.4 for the *Taia* and *Taib* instances. For *Sko*, only PMTS (Czapiński, 2013) has results reported, and they are of comparable quality with those from the state-of-the-art CPU metaheuristics.

Comparison for the QBAP

Ideally, a fair way to evaluate a heuristic is by comparing it against other heuristics only. However, in the absence of other published heuristics to solve QBAP, we compare our PMITS against the solutions found by CPLEX for reference using the models presented in Section 4.2.2. CPLEX can provide upper/lower bounds to the optimal solution. This has a practical purpose of showing how one can benefit from using a heuristic instead of an exact solver, such as the speedup provided and the difference in solution quality. The results are shown in Table 4.5. Wall-clock times are reported in minutes. We show the best solution found by CPLEX with any model and the best and average solution found by our heuristic for 20 runs. Each solution shown in the table found by the solver is the optimal solution for the problem when the time is under 60 minutes, or the best upper bound found when the run stopped. In most of the larger instances, *n.s.* is displayed in the table meaning that no solution is found within the time limit. PMITS finds the BKS in all 31 instances. We note that in instances with $n < 20$ of *Taia* and $n < 60$ of *Taib*, PMITS finds the optimal solution in all 20 runs.

Comparison for the BiQAP

We compare our PMITS for the Biquadratic QAP to the GRASP by Mavridou et al. (1998), which was kindly provided by the authors so that both methods could be run in the same computation environment. In our experiments, both algorithms were run 20 times. GRASP parameters were set as in the original paper. We limited it to a maximum of 100 iterations per run. The results using both methods are presented in Table 4.6. Wall-clock times are reported in minutes. All BKS are found only by PMITS. The APD reported is from the average solution of all runs to the BKS. For the PMITS, we show two different APDs. The first is from the solution found after the algorithm stopped using its regular stopping criteria (convergence and time), while the second is the APD at the moment that PMITS spent the

Table 4.3 – Comparison of MITS with state-of-the-art sequential metaheuristics for the QAP

Instance	*BKS*	*JRG-DivTS*		*ITS*		*TLBO-RTS*		*BMA*		*MITS*	
		APD	*Time*	*APD*	*Time*	*APD*	*Time*	*APD*	*Time*	*APD*	*Time*
tai20a	703,482	**0.00**	0.2	**0.00**	0.0	**0.00**	5.2	**0.00**	0.0	**0.00**	0.6
tai25a	1,167,256	**0.00**	0.6	**0.00**	0.1	**0.00**	8.3	**0.00**	0.0	**0.00**	2.1
tai30a	1,818,146	**0.00**	1.3	**0.00**	0.2	**0.00**	12.1	**0.00**	0.0	**0.00**	4.9
tai35a	2,422,002	**0.00**	4.4	**0.00**	0.5	**0.00**	16.7	**0.00**	0.0	**0.00**	17.4
tai40a	3,139,370	0.22	5.2	0.22	1.3	0.07	57.5	0.06	8.1	**0.05**	60.0
tai50a	4,938,796	0.73	10.2	0.41	5.5	0.55	127.6	**0.13**	42.0	0.19	60.0
tai60a	7,205,962	0.72	25.7	0.45	12.5	0.64	128.5	**0.14**	67.5	0.32	60.0
tai80a	13,499,184	0.87	52.7	0.48	60.0	0.77	244.8	**0.43**	65.8	0.55	60.0
tai100a	21,043,560	0.90	142.1	**0.35**	200.0	1.08	385.7	0.44	44.1	0.57	60.0
tai20b	122,455,319	**0.00**	0.2	**0.00**	0.0	**0.00**	0.3	**0.00**	0.0	**0.00**	0.0
tai25b	344,355,646	**0.00**	0.5	**0.00**	0.0	**0.00**	0.3	**0.00**	0.0	**0.00**	0.1
tai30b	637,117,113	**0.00**	1.3	0.01	0.0	**0.00**	1.0	**0.00**	0.0	**0.00**	17.6
tai35b	283,315,445	**0.00**	2.4	0.02	0.1	**0.00**	1.5	**0.00**	0.0	**0.00**	0.7
tai40b	637,250,948	**0.00**	3.2	0.01	0.2	**0.00**	2.3	**0.00**	0.0	**0.00**	2.1
tai50b	458,821,517	**0.00**	8.8	0.02	0.5	**0.00**	4.4	**0.00**	1.2	**0.00**	8.5
tai60b	608,215,054	**0.00**	17.1	0.04	1.1	**0.00**	7.6	**0.00**	5.2	**0.00**	58.4
tai80b	818,415,043	0.01	58.2	0.23	3.0	0.01	18.8	**0.00**	31.3	**0.00**	60.0
tai100b	1,185,996,137	0.06	118.9	0.14	6.7	0.05	36.1	**0.00**	13.6	**0.00**	60.0
sko42	15,812	**0.00**	4.0	**0.00**	0.2	**0.00**	7.7	**0.00**	0.0	**0.00**	1.0
sko49	23,386	0.01	9.6	0.01	0.3	0.01	4.1	**0.00**	0.0	**0.00**	48.7
sko56	34,458	0.00	13.2	0.02	0.7	0.00	18.5	**0.00**	0.0	**0.00**	18.2
sko64	48,498	**0.00**	22.0	0.01	1.5	0.00	27.5	**0.00**	0.0	**0.00**	43.0
sko72	66,256	0.01	38.0	0.06	3.0	0.02	39.2	**0.00**	3.5	0.01	60.0
sko81	90,998	0.02	56.4	0.06	6.0	0.02	56.6	**0.00**	4.3	0.02	60.0
sko90	115,534	0.03	89.6	0.07	12.0	0.03	79.4	**0.00**	15.3	0.01	60.0
sko100a	152,002	0.03	129.2	0.09	30.0	0.04	109.5	**0.00**	22.3	0.05	60.0
sko100b	153,890	0.01	106.6	0.06	30.0	0.03	109.4	**0.00**	6.5	0.02	60.0
sko100c	147,862	0.01	126.7	0.06	30.0	0.02	109.6	**0.00**	12.0	0.01	60.0
sko100d	149,576	0.03	123.5	0.09	30.0	0.04	109.3	**0.01**	20.9	0.03	60.0
sko100e	149,150	0.01	108.8	0.07	30.0	0.02	109.7	**0.00**	11.9	0.01	60.0
sko100f	149,036	0.02	110.3	0.08	30.0	0.02	109.5	**0.00**	23.0	0.06	60.0
Average		0.12	44.9	0.10	16.0	0.11	62.9	**0.04**	12.9	0.06	38.2

same time as the GRASP for a fair comparison between both methods. The results show that PMITS outperforms GRASP by finding solutions at 0.07% from BKS compared to 0.40% for the GRASP. We also notice that by letting the algorithm run longer the average APD is reduced to 0.03%, which indicates that solutions keep improving by our heuristic. In fact, only instances with up to 14 facilities are stopped by the convergence criteria and instances with up to 18 facilities have the BKS found in all 10 runs. This shows how difficult it is to solve the BiQAP for the conditions considered in the set of instances tested.

Comparison for the QSAP

For the same reasons as for the QBAP, we conducted tests on the two sets of instances for the QSAP using our PMITS heuristic and CPLEX with the models presented in Section 4.2.4.

Table 4.4 – Comparison of PMITS with state-of-the-art parallel metaheuristics for the QAP

Instance	*BKS*	*CPTS*		*TLBO-RTS*		*PHA*		*MSH-QAP*		*PHYSH-QAP*		*BLS-OpenMP*		*PABC-QAP*		*PMITS*	
		10 cores		*40-50 cores*		*46 cores*		*64 cores*		*64 cores*		*8-16 cores*		*255 cores*		*40 cores*	
		APD	*Time*	*APD*	*Time*	*APD*	*Time*	*APD*	*Time*	*APD*	*Time*	*APD*	*Time*	*APD*	*Time*	*APD*	*Time*
tai20a	703,482	**0.00**	0.1	–	–	**0.00**	0.4	**0.00**	1.0	–	–	**0.00**	0.1	**0.00**	0.0	**0.00**	0.0
tai25a	1,167,256	**0.00**	0.3	–	–	**0.00**	0.6	**0.00**	1.3	–	–	**0.00**	0.1	**0.00**	0.1	**0.00**	0.1
tai30a	1,818,146	**0.00**	1.6	–	–	**0.00**	1.0	**0.00**	1.5	–	–	**0.00**	0.2	**0.00**	0.2	**0.00**	0.2
tai35a	2,422,002	**0.00**	2.3	**0.00**	18.3	**0.00**	1.3	**0.00**	1.8	–	–	**0.00**	0.3	**0.00**	0.0	**0.00**	0.5
tai40a	3,139,370	0.15	3.5	**0.00**	29.0	**0.00**	10.6	0.26	30.0	0.03	2.9	**0.00**	32.2	**0.00**	4.0	**0.00**	60.0
tai50a	4,938,796	0.44	10.3	0.36	55.0	**0.00**	12.7	0.17	37.5	0.13	4.4	**0.00**	68.2	0.31	19.8	**0.00**	60.0
tai60a	7,205,962	0.48	26.4	0.41	95.3	**0.00**	19.6	0.27	45.0	0.24	4.9	**0.00**	107.9	0.45	34.0	0.08	60.0
tai80a	13,499,184	0.69	94.8	0.87	239.5	0.64	40.0	0.53	60.0	0.46	5.0	0.50	235.9	0.83	173.0	**0.35**	60.0
tai100a	21,043,560	0.63	261.2	0.64	483.3	0.58	71.9	0.41	75.0	0.39	5.0	0.65	448.5	0.64	335.6	**0.34**	60.0
tai20b	122,455,319	**0.00**	0.1	–	–	**0.00**	0.4	**0.00**	1.0	**0.00**	0.0	**0.00**	0.1	**0.00**	0.0	**0.00**	0.0
tai25b	344,355,646	**0.00**	0.4	–	–	**0.00**	0.6	**0.00**	1.3	**0.00**	0.0	**0.00**	0.0	**0.00**	0.0	**0.00**	0.0
tai30b	637,117,113	**0.00**	1.2	–	–	**0.00**	0.8	**0.00**	1.5	**0.00**	0.0	**0.00**	0.2	**0.00**	0.1	**0.00**	1.0
tai35b	283,315,445	**0.00**	5.5	**0.00**	22.4	**0.00**	1.1	**0.00**	1.8	**0.00**	0.0	**0.00**	0.3	**0.00**	0.0	**0.00**	0.0
tai40b	637,250,948	**0.00**	4.5	–	–	**0.00**	1.6	**0.00**	2.0	**0.00**	0.0	**0.00**	0.3	**0.00**	0.2	**0.00**	0.1
tai50b	458,821,517	**0.00**	13.8	–	–	**0.00**	5.8	**0.00**	3.0	**0.00**	0.1	**0.00**	0.7	**0.00**	1.6	**0.00**	0.3
tai60b	608,215,054	**0.00**	30.4	–	–	**0.00**	9.5	**0.00**	3.2	**0.00**	0.2	**0.00**	18.6	**0.00**	0.7	**0.00**	6.9
tai80b	818,415,043	**0.00**	110.9	**0.00**	239.0	**0.00**	27.7	**0.00**	4.0	**0.00**	0.7	**0.00**	218.1	**0.00**	8.0	**0.00**	60.0
tai100b	1,185,996,137	**0.00**	241.0	**0.00**	508.2	**0.00**	42.5	**0.00**	5.0	**0.00**	2.4	**0.00**	160.8	**0.00**	164.7	**0.00**	60.0
sko42	15,812	**0.00**	5.3	–	–	**0.00**	1.6	**0.00**	2.1	–	–	**0.00**	0.4	**0.00**	0.2	**0.00**	0.1
sko49	23,386	**0.00**	11.4	**0.00**	54.1	**0.00**	4.0	**0.00**	2.5	–	–	**0.00**	0.6	**0.00**	7.5	**0.00**	13.3
sko56	34,458	**0.00**	21.0	**0.00**	81.6	**0.00**	16.2	**0.00**	2.8	**0.00**	0.0	**0.00**	0.8	**0.00**	0.9	**0.00**	0.7
sko64	48,498	**0.00**	42.9	**0.00**	119.3	**0.00**	23.1	**0.00**	3.2	**0.00**	0.0	**0.00**	1.2	**0.00**	2.4	**0.00**	2.4
sko72	66,256	**0.00**	69.6	**0.00**	170.8	**0.00**	33.6	**0.00**	3.6	**0.00**	0.2	**0.00**	1.8	**0.00**	7.0	**0.00**	60.0
sko81	90,998	**0.00**	121.4	–	–	**0.00**	39.9	**0.00**	4.1	**0.00**	0.4	**0.00**	2.4	**0.00**	15.2	**0.00**	60.0
sko90	115,534	**0.00**	193.7	**0.00**	342.8	**0.00**	40.5	**0.00**	4.5	**0.00**	1.7	**0.00**	3.2	**0.00**	14.7	**0.00**	40.9
sko100a	152,002	**0.00**	304.8	**0.00**	594.3	**0.00**	41.7	**0.00**	75.0	**0.00**	2.2	**0.00**	29.8	0.01	170.5	0.01	60.0
sko100b	153,890	**0.00**	309.6	0.01	482.6	**0.00**	42.3	**0.00**	75.0	**0.00**	0.9	**0.00**	8.5	**0.00**	127.7	**0.00**	60.0
sko100c	147,862	**0.00**	316.1	**0.00**	508.5	**0.00**	42.2	**0.00**	75.0	**0.00**	1.3	**0.00**	4.3	**0.00**	48.3	**0.00**	60.0
sko100d	149,576	**0.00**	309.8	0.01	509.4	**0.00**	41.9	**0.00**	75.0	**0.00**	1.1	**0.00**	12.9	**0.00**	45.9	0.01	58.5
sko100e	149,150	**0.00**	309.1	0.01	614.5	**0.00**	42.5	**0.00**	75.0	**0.00**	0.8	**0.00**	4.3	**0.00**	42.1	**0.00**	60.0
sko100f	149,036	**0.00**	310.3	0.01	482.6	**0.00**	42.0	**0.00**	75.0	**0.00**	1.7	**0.00**	17.1	**0.00**	28.3	0.01	60.0
Average		0.08	101.0	0.12	282.5	0.04	21.3	0.05	24.2	0.05	1.4	0.04	44.5	0.07	40.4	**0.03**	31.1

The results are presented in Table 4.7. For set 1, which contains smaller instances, the exact method proved the optimal solution for 19 out of 21 instances. The PMITS not only finds all BKS, including the two instances without proven optimality, but it also presents a very high consistency by finding the BKS in all runs for all instances. All that in an average time of less than a tenth of minute, compared to approximately nine minutes for the exact models. The set 2 is harder to solve due to the larger number of facilities and sites. None of the 20 instances has the optimality proven by the exact method, although in two occasions the BKS are equally found in both methods. For instances with up to $|\mathcal{N}| = 125$, the PMITS always stops when reaching the convergence stopping criteria. In larger instances, although the heuristic has not converged, in all of them the same solution is found in all 20 runs.

Comparison for the GQAP

We compare next the proposed PMITS with other heuristics available in the literature for the QSAP. They are the memetic heuristic (Memetic) by Cordeau et al. (2006), the GRASP (GRASP-PR) by Mateus et al. (2011), and the tabu search (TS) by McKendall and Li (2017). Following the procedure used in the two latter papers, we added a new stopping criterion so that the PMITS stops the search right after finding the BKS reported in the other works. Table 4.8 shows the results obtained for each instance tested with wall-clock times reported in minutes. The times for the other heuristics are those reported in their works. We highlight

Table 4.5 – Comparison of PMITS with the best results found by CPLEX for the QBAP

Instance	*CPLEX*		*PMITS*		
	Solution	*Time (min)*	*Best sol.*	*Avg. sol.*	*Time (min)*
tai10a	**4,256**	0.1	**4,256**	4,256	0.0
tai12a	**4,756**	0.2	**4,756**	4,756	0.2
tai15a	**4,757**	0.9	**4,757**	4,757	25.3
tai17a	**4,704**	18.5	**4,704**	4,704	60.0
tai20a	5,200	> 60.0	**5,096**	5,122.6	60.0
tai25a	5,880	> 60.0	**5,328**	5,474.1	60.0
tai30a	6,992	> 60.0	**5,952**	5,999.3	60.0
tai35a	8,613	> 60.0	**6,120**	6,254	60.0
tai40a	9,306	> 60.0	**6,370**	6,475.6	60.0
tai50a	9,604	> 60.0	**6,873**	6,899.2	60.0
tai60a	9,702	> 60.0	**7,134**	7,170.6	60.0
tai80a	*n.s.*		**7,695**	7,731.3	60.0
tai100a	*n.s.*		**8,036**	8,093	60.0
tai12b	**4,371,380**	0.6	**4,371,380**	4,371,380	0.0
tai15b	**22,204,329**	0.7	**22,204,329**	22,204,329	0.0
tai20b	**17,132,916**	> 60.0	**17,132,916**	17,132,916	0.0
tai25b	31,797,414	> 60.0	**17,166,072**	17,166,072	0.1
tai30b	49,847,879	> 60.0	**25,462,115**	25,462,115	0.3
tai35b	16,153,962	> 60.0	**7,159,056**	7,159,056	7.1
tai40b	24,261,283	> 60.0	**12,566,129**	12,566,129	14.9
tai50b	13,002,408	> 60.0	**5,180,968**	5,180,968	30.0
tai60b	15,427,179	> 60.0	**5,800,564**	5,837,290.4	60.0
tai80b	*n.s.*		**3,335,332**	3,384,604.8	60.0
tai100b	*n.s.*		**4,188,992**	4,233,119.1	60.0
sko42	100	> 60.0	**50**	53.5	32.4
sko49	110	> 60.0	**60**	63	17.1
sko56	120	> 60.0	**70**	76	24.2
sko64	130	> 60.0	**80**	82	48.0
sko72	140	> 60.0	**90**	91.5	51.0
sko81	*n.s.*		**100**	103.5	39.1
sko90	*n.s.*		**100**	109.5	43.5
sko100a	*n.s.*		**110**	119	36.6
sko100b	*n.s.*		**120**	122	48.1
sko100c	*n.s.*		**110**	119.5	15.5
sko100d	*n.s.*		**120**	120	40.5
sko100e	*n.s.*		**120**	120.5	57.0
sko100f	*n.s.*		**110**	118	29.3

that the TS by McKendall and Li (2017) was stopped in the instance *30-20-95* in a different solution, 0.27% far from the BKS. The difficulty to solve this instance comes from the tight capacity available in the sites, which makes the search for feasible solutions longer using random constructive heuristics. Around a third of the time spent by PMITS on this instance was trying to generate the initial population. Although a direct comparison of running time between the different heuristics is not possible since they were run in different environments, we can see how effective the proposed PMITS is. Excluding the hard *30-20-95* instance, the average time for our heuristic to find the BKS is less than one second.

Table 4.6 – Comparison of PMITS with GRASP by Mavridou et al. (1998) for the BiQAP

Instance	*BKS*	*GRASP*		*PMITS*			
		APD	*Time*	*APD*	*Time*[1]	*APD*	*Time*[2]
$\|\mathcal{N}\| = 10$	239,980	0.15	0.0	0.00	0.0	**0.00**	0.0
$\|\mathcal{N}\| = 12$	478,278	0.63	0.0	0.00	0.0	**0.00**	0.4
$\|\mathcal{N}\| = 14$	915,566	0.46	0.1	0.00	0.1	**0.00**	4.2
$\|\mathcal{N}\| = 16$	1,572,146	0.58	0.1	0.05	0.1	**0.00**	60.0
$\|\mathcal{N}\| = 18$	2,523,592	0.53	0.3	0.05	0.3	**0.00**	60.0
$\|\mathcal{N}\| = 20$	3,846,588	0.61	0.5	0.16	0.6	**0.02**	60.0
$\|\mathcal{N}\| = 22$	5,681,512	0.55	1.0	0.16	1.0	**0.05**	60.1
$\|\mathcal{N}\| = 24$	8,084,696	0.42	1.6	0.14	1.8	**0.05**	60.1
$\|\mathcal{N}\| = 26$	11,144,442	0.39	2.7	0.13	2.9	**0.06**	60.2
$\|\mathcal{N}\| = 28$	15,091,044	0.33	4.2	0.09	4.7	**0.04**	60.3
$\|\mathcal{N}\| = 30$	19,835,460	0.29	6.3	0.07	7.0	**0.03**	60.4
$\|\mathcal{N}\| = 32$	25,768,226	0.27	9.6	0.05	10.1	**0.04**	60.8
$\|\mathcal{N}\| = 34$	32,885,928	0.20	14.2	0.03	15.8	**0.02**	61.2
$\|\mathcal{N}\| = 36$	41,295,160	0.24	19.6	0.08	21.3	**0.07**	61.1
Average		0.40	4.3	0.07	4.7	0.03	47.8

[1] Stopping criteria is the running time from GRASP

[2] Regular stopping criteria (50% convergence or one-hour time limit)

4.6 Conclusions

In this paper, we have presented a simple yet effective parallel memetic iterated tabu search (PMITS) algorithm, adapted to solve the Quadratic Assignment Problem (QAP) and four of its variants, i.e., the Quadratic Bottleneck Assignment Problem (QBAP), the Biquadratic Assignment Problem (BiQAP), the Quadratic Semi-Assignment Problem (QSAP), and the Generalized Quadratic Assignment Problem (GQAP). Although these variants have many practical applications, some of their literature is outdated compared to the advances recently achieved for the QAP. We have provided an important survey of the literature and algorithmic structures for these problems.

The PMITS was developed following the successful diversification-intensification approach for QAPs, where solutions evolve using a memetic algorithm to generate new solutions from combinations of existing ones in the current population, then being improved by one or two local searches, depending on the problem, and by a tabu search whenever new local optima are found. Although the algorithm exploits an iterated tabu search, which is well-known to work well on QAP, we modify some classical frameworks creating new ones – such as the new skewed crossover, the modified tabu list structure, the improved acceptance criteria for the tabu search, and the new long-term memory using a map to speedup the search process – to improve its effectiveness for the QAP. We also extend the objective function evaluation speedup formulas for the QAP to its variants. The algorithm was set for each individual problem through computational experiments using sets of benchmark instances found in literature, among other instances created here. The results of the experiments show that PMITS is

Table 4.7 – Comparison of PMITS with the best results found by CPLEX for the QSAP

Instance	*CPLEX*		*PMITS*		
	Sol.	*Time (min)*	*Best sol.*	*Avg. sol.*	*Time (min)*
20-15-35	**1,599,473**	0.4	**1,599,473**	1,599,473	0.0
20-15-55	**1,427,052**	0.2	**1,427,052**	1,427,052	0.0
20-15-75	**1,648,679**	0.4	**1,648,679**	1,648,679	0.0
30-06-95	**5,486,902**	0.1	**5,486,902**	5,486,902	0.0
30-07-75	**4,834,397**	0.1	**4,834,397**	4,834,397	0.0
30-08-55	**4,484,813**	0.2	**4,484,813**	4,484,813	0.0
30-10-65	**3,649,165**	0.2	**3,649,165**	3,649,165	0.0
30-20-35	**3,351,755**	15.0	**3,351,755**	3,351,755	0.0
30-20-55	**3,247,260**	6.6	**3,247,260**	3,247,260	0.0
30-20-75	**3,301,384**	60.0	**3,301,384**	3,301,384	0.0
30-20-95	**2,941,907**	3.0	**2,941,907**	2,941,907	0.0
35-15-35	**4,533,539**	3.2	**4,533,539**	4,533,539	0.0
35-15-55	**4,220,924**	2.0	**4,220,924**	4,220,924	0.0
35-15-75	**5,620,789**	60.0	**5,620,789**	5,620,789	0.0
35-15-95	**4,555,240**	5.7	**4,555,240**	4,555,240	0.0
40-07-75	**8,347,601**	0.8	**8,347,601**	8,347,601	0.0
40-09-95	**7,107,977**	1.9	**7,107,977**	7,107,977	0.0
40-10-65	**7,509,269**	2.2	**7,509,269**	7,509,269	0.0
50-10-65	**11,795,583**	2.7	**11,795,583**	11,795,583	0.0
50-10-75	**10,107,391**	1.5	**10,107,391**	10,107,391	0.0
50-10-95	**11,882,812**	15.6	**11,882,812**	11,882,812	0.0
50-C1	**1,022,084**	60.0	**1,022,084**	1,022,084	0.0
50-C10	1,124,895	60.0	**1,123,607**	1,123,607	0.0
50-C25	1,293,354	60.0	**1,292,223**	1,292,223	0.0
50-C50	**1,573,474**	60.0	**1,573,474**	1,573,474	0.0
75-C1	2,045,813	60.0	**1,993,829**	1,993,829	0.0
75-C10	2,232,142	60.0	**2,195,019**	2,195,019	0.0
75-C25	2,581,945	60.0	**2,530,699**	2,530,699	0.0
75-C50	3,124,028	60.0	**3,089,736**	3,089,736	0.0
100-C1	3,731,562	60.0	**3,490,341**	3,490,341	0.1
100-C10	4,469,237	60.0	**3,836,299**	3,836,299	0.1
100-C25	5,085,661	60.0	**4,412,117**	4,412,117	0.0
100-C50	6,083,200	60.0	**5,370,205**	5,370,205	0.0
125-C1	5,358,958	60.0	**4,881,972**	4,881,972	0.6
125-C10	6,261,164	60.0	**5,375,051**	5,375,051	0.7
125-C25	6,713,149	60.0	**6,196,362**	6,196,362	0.7
125-C50	7,902,849	60.0	**7,564,714**	7,564,714	0.8
150-C1	14,631,247	60.0	**6,930,942**	6,930,942	7.5
150-C10	15,934,010	60.0	**7,625,357**	7,625,357	7.2
150-C25	18,104,789	60.0	**8,782,100**	8,782,100	58.0
150-C50	21,721,652	60.0	**10,707,271**	10,707,271	57.5
Average	5,918,271.3	33.7	4,918,227.2	4,918,227.2	3.2

among the best performing heuristics developed for the QAP, specially on the larger Taia instances, which are undoubtedly the hardest benchmark instances to solve from QAPLIB, and significantly outperforms the state-of-the-art methods for the other four problems.

Table 4.8 – Comparison of PMITS with state-of-the-art heuristics for the GQAP

Instance	*BKS*	*Memetic*	*GRASP-PR*	*TS*	*PMITS*
		Time	*Time*	*Time*	*Time*
20-15-35	1,471,896	1.6	0.1	0.0	0.0
20-15-55	1,723,638	1.7	0.0	0.0	0.0
20-15-75	1,953,188	1.7	0.0	4.2	0.0
30-06-95	5,160,920	1.9	0.0	0.0	0.0
30-07-75	4,383,923	2.6	0.1	0.0	0.0
30-08-55	3,501,695	1.6	0.0	0.0	0.0
30-10-65	3,620,959	3.5	2.0	0.1	0.0
30-20-35	3,379,359	9.4	1.3	32.8	0.0
30-20-55	3,593,105	7.7	0.4	24.6	0.0
30-20-75	4,050,938	8.7	0.7	0.0	0.0
30-20-95	5,710,645	87.2	9050.3	0.2[1]	9.0
35-15-35	4,456,670	7.6	5.1	26.8	0.0
35-15-55	4,639,128	6.4	0.4	0.0	0.0
35-15-75	6,301,723	6.6	1.1	0.0	0.0
35-15-95	6,670,264	14.4	24.2	0.0	0.0
40-07-75	7,405,793	3.0	1.0	0.0	0.0
40-09-95	7,667,719	19.0	7.0	0.1	0.1
40-10-65	7,265,559	4.0	0.3	0.0	0.0
50-10-65	10,513,029	8.4	0.4	0.0	0.0
50-10-75	11,217,503	10.1	22.5	0.1	0.0
50-10-95	12,845,598	20.9	1.5	0.4	0.0
Average		10.9	434.2	4.2	0.4

[1] Avg. time reported to find the solution 5,726,530 (APD of 0.27% from BKS)

Chapter 5

Robotic mobile fulfillment system with pod repositioning for energy saving

Résumé

Le système à robots mobile permet un repositionnement facile d'inventaire en retournant des pods à différents endroits après qu'ils ont été demandés. Une meilleure organisation des pods mène à un processus de prélèvement économe en énergie, car ils peuvent être garés plus près d'où ils seront demandés au futur. Quand une commande arrive, nous devons décider quelle station de prélèvement s'en occupera (affectation de commande), quelle pod contenant les produits demandés doit être amenée à cette station (sélection de pods), et vers quel endroit de stockage ces pods doivent retourner après le prélèvement (repositionnement de pods). Dans ce chapitre, ces problèmes sont résolus de manière intégrée en utilisant une stratégie de prélèvement par vague, où les décisions sont prises périodiquement. Le problème est d'abord modélisé pour le cas où les demandes futures sont connues. Cependant, étant donné que des informations complètes sur les demandes futures sont rarement disponibles dans la pratique, nous proposons d'autres modèles pour le résoudre, soit lorsqu'aucune information sur les demandes futures existent (*approche myope*) ou lorsque les demandes sont incertaines (*approche stochastique*). Une matheuristique de recherche locale est présentée pour résoudre des instances de taille réelle. Nous mesurons la réduction de la consommation d'énergie lorsque les décisions de repositionnement de pods sont intégrées aux autres décisions et lorsqu'un schéma d'échantillonnage est utilisé pour représenter les demandes futures. Cette mesure est faite par des expériences pour des instances avec des caractéristiques réelles trouvées dans les RMFS. Nos résultats attestent de l'efficacité de la considération des données stochastiques des demandes lors de la planification des problèmes opérationnels résolus.

Summary

The robotic mobile fulfillment system allows easy inventory repositioning by returning pods to different locations after being requested. A better inventory arrangement leads to an energy-efficient picking process since pods can be parked closer to where they will be requested next. After an order arrives, we have to decide which picking station will deal with it (order assignment), which pod containing the products requested should be brought to this station (pod selection), and to which storage location these pods should return after picking (pod repositioning). In this chapter, we solve these problems in an integrated manner using a wave picking strategy, where decisions are made periodically. They are first modeled for when future demands are known. Since full information about future demands is seldom available in practice, we propose different models to solve it, such as when no information about the future demands is available (*myopic approach*) or when demands are uncertain (*stochastic approach*). A local search matheuristic is presented to solve real-size instances. We measure the energy consumption reduction when pod repositioning is integrated with the other decisions and when a sampling scheme represents future demands through computational experiments for instances with real characteristics found in an RMFS. Our results attest to the effectiveness of considering stochastic data for demands when planning the operational problems solved.

5.1 Introduction

Although many distribution center operations are still very labor-intensive, mainly due to the advent of e-commerce, a growing effort to automate some of these processes is observed in the last decades (Azadeh et al., 2019). Automated warehousing systems work with manual picking stations as a product-to-picker system. A growing category of automated systems is the robotic mobile fulfillment system (RMFS), where mobile robots can lift movable inventory pods and bring them directly to stationary human pickers in fixed stations located around the storage area (Boysen et al., 2019a). This system was just recently popularized (Guizzo, 2008; Mountz et al., 2008; Wurman et al., 2008). Since then, many providers have entered the mobile robots market and many large online retailers have switched the entirely manual picking operations in their distribution centers to an RMFS (Boysen et al., 2019a; Lamballais et al., 2020; Weidinger et al., 2018; Zou et al., 2018).

In addition to the intrinsic advantages associated with automation, an RMFS provides increased flexibility and scalability due to the ease of adding and removing pods and robots in the system and repositioning inventory in the storage area. They also require a relatively low investment cost, even for a large fleet of robots compared to other automated systems. Picking rates can be more than double in an RMFS compared to manual warehouses since they eliminate the unproductive walking time of pickers (Roodbergen and Vis, 2009). Due to the lifting capacity of robots, the RMFS is efficient when used in warehouses containing

several small and lightweight items, which makes it a perfect choice for e-commerce (Azadeh et al., 2019).

Early studies dealing with operational problems in the RMFS literature focused on optimizing performance measures related to the maximization of order throughput (Lamballais et al., 2017; Zou et al., 2017). Although optimizing this measure is important in scenarios where a very tight delivery deadline has to be met, in reality, most applications allow picking tasks to wait if this results in a more efficient overall picking process. For example, orders with standard shipping can be delayed over orders with priority shipping. Minimizing operation costs is as important as reducing the picking time in periods of low demand (Makris et al., 2006). These costs are usually associated with the number of pod visits to stations, which is another common measure optimized (Aldarondo and Bozer, 2020; Jiang et al., 2020; Xiang et al., 2018), or the distance traveled by robots (Weidinger et al., 2018). With many robots running simultaneously in a typical warehouse, the total energy consumed is significantly high. Using energy consumption as a performance measure can balance pod visits and distances traveled by robots since energy consumption is directly related to both of them.

A common strategy to improve order picking efficiency is to process orders in batches using wave picking (Çeven and Gue, 2017), which is adopted in real-world RMFS (Zhuang et al., 2021). At the beginning of a wave, the current state of the system is known (number and capacity of operating picking stations, current set of orders in the backlog, and position of pods within the storage area). Then, several operational decisions are made, such as the assignment of orders to stations (*order assignment problem*, OAP), the selection of pods to be carried to the stations in the current wave (*pod selection problem*, PSP), and the location they should return to after picking (*pod repositioning problem*, PRP). Although these problems are known to be interrelated (Merschformann et al., 2019), only the OAP and the PSP have been extensively investigated in an integrated manner (Jiang et al., 2020; Valle and Beasley, 2021; Xiang et al., 2018; Xie et al., 2021; Zhuang et al., 2021). In wave picking, the integration of PRP decisions with the OAP and the PSP can lead to a better arrangement of inventory in the storage area for future waves. However, no study could be found that integrates decisions for all these problems, being one of this research main objectives.

We investigate different approaches to solve the PRP in an integrated manner with the OAP and the PSP in an RMFS using a wave picking strategy. The objective is to reduce the total energy consumed by robots carrying the pods around the storage area to meet the demands. We first consider an ideal scenario where the demands of future waves are already known to model the integrated problem as a dynamic integer non-linear programming (DINLP) problem. We acknowledge that it is unrealistic to dispose of full information about future demands in most situations. Therefore, we derive new methods to solve the integrated problem by planning waves individually. Given the information available at the beginning of each wave, in the *sequential* approach, the problem is solved using a two-phase procedure. In the first

phase, the OAP and the PSP decisions are made using an integer linear programming (ILP) model adapted from the literature (Valle and Beasley, 2021; Xie et al., 2021). Then, in the second phase, the PRP is solved using the "nearest rule", which assigns pods to the nearest available locations (Merschformann et al., 2019). The sequential approach is used as a benchmark to evaluate the benefits of integrating decisions for the PRP with the OAP and the PSP. The integrated problem, in which all three problems are solved together for a wave, is modeled as an integer non-linear programming (INLP) problem. Since no information about future demands is available, we call it the *myopic* approach. Alternatively, the integrated problem is also solved considering that future demands can be predicted with uncertainty. The *stochastic* approach considers that scenarios for future demands can be sampled from an ABC distribution function commonly used in the warehousing literature, as it accounts for the skewness of demands (Caron et al., 1998). This approach is modeled using a two-stage stochastic programming model and is solved with a Benders decomposition scheme. The first stage solves the integrated problem for the current wave, and the second stage solves for the upcoming wave represented by the sampled scenarios. We observe that the models for the myopic and stochastic approaches are too heavy to deal with real-case instances. Thus, we present a local search matheuristic where the sequential approach solution is improved by a simple local search to approximate optimal solutions for the other approaches.

Extensive computational experiments on instances generated based on the data commonly used in the RMFS literature are performed to assess the impact on energy consumption when planning a wave using all three approaches, i.e., sequential, myopic, and stochastic. Their solutions are also compared against the DINLP solutions to observe how far they are from the best possible case. We further extend our analysis to evaluate the energy consumption for when the number of pod visits is minimized and when we can delay picks until more orders arrive before planning the waves.

This paper is structured as follows. Section 5.2 provides a literature review on the relevant papers that solved the OAP, PSP, and PRP. Section 5.3 describes the RMFS considered, the decision problems, and how the energy consumption of the robots is estimated. Section 5.4 introduces the mathematical models for the approaches considered. Section 5.5 presents the local search matheuristic used to solve large instances of the integrated problem. Section 5.6 reports the computational results for the different approaches and situations analyzed. Finally, Section 5.7 presents the concluding remarks of this paper.

5.2 Literature review

Nearly all studies dealing with an RMFS at the operational level, to some extent, use a solution strategy to the three problems considered (OAP, PSP, and PRP). We summarize the relevant studies that investigate these problems, focusing on those that study the PRP or at

least two of the three problems considered. They are referenced in Table 5.1, showing the problems investigated, the picking strategy used (real-time, wave, or single solution), and the performance measures considered. More details about the RMFS are found in the reviews of Azadeh et al. (2019); Boysen et al. (2019a), and Jaghbeer et al. (2020).

Table 5.1 – Studies that investigate the OAP, PSP, or PRP

Reference	OAP	PSP	PRP	Picking strategy	Performance measure
Weidinger et al. (2018)			✓	Single solution	Distance traveled
Xiang et al. (2018)	✓	✓		Single solution	Pod visits
Merschformann et al. (2019)	✓	✓	✓	Real-time	Order throughput rate
Aldarondo and Bozer (2020)		✓	✓	—	Pod visits
Li et al. (2020)			✓	Real-time	Energy consumption
Jiang et al. (2020)	✓	✓		Wave	Pod visits
Rimélé et al. (2021)	✓	✓	✓	Real-time	Cycle and travel times
Xie et al. (2021)	✓	✓		Real-time, Wave	Pod visits
Valle and Beasley (2021)	✓	✓		Single solution	Pod visits
Zhuang et al. (2021)	✓	✓		Wave	Pods movements
Our study	✓	✓	✓	Wave	Energy consumption

The OAP and the PSP are the problems mostly investigated together on order picking in the RMFS literature. Xiang et al. (2018) solve the integrated OAP–PSP after replenishment decisions are made using an ILP model. A heuristic procedure is suggested to solve the integrated problem considering product correlations and order associations. A variable neighborhood search (VNS) is used to search for improved solutions by exchanging orders between batches. Later, Jiang et al. (2020) integrate replenishment decisions made in waves with the OAP and the PSP. A divide-and-conquer paradigm is used to generate an initial solution for the problem. Then, another VNS is used to improve it. Xie et al. (2021) also integrate the OAP and the PSP by proposing several ILPs where orders are allowed or forbidden to be split among different batches or periods. Although they consider real time picking using a simulation framework, the problem is solved periodically, when some jobs are finished at the stations, such that orders are assigned in batches to the stations. A heuristic is proposed to accelerate the computational time. The PRP is solved in the simulations using a policy that positions pods in the nearest available location. They show that their integrated approach significantly reduces the number of pod visits to stations compared to when OAP and PSP decisions are made sequentially, such as in Merschformann et al. (2019). Valle and Beasley (2021) also integrate OAP and PSP decisions using an ILP. Many additional constraints are proposed, such as adding picks from different pod sides, allocating a single pod to multiple stations, and balancing the workload among pickers. Two heuristics are proposed to solve the integrated problem. The first is based on the assignment of batches to one station at a time. The second is based on fixing parts of the decision variables and solving the resulting sub-problem (partial integer optimization). After this problem is solved, they also solve the pod sequencing problem. Finally, Zhuang et al. (2021) consider the OAP and the PSP and integrate them

with the order and pod sequencing problems. As most of the other mentioned studies, the objective is to minimize the number of pods visiting stations, although they consider a slightly different measure to also count the number of movements between stations. The integrated problem is modeled using an ILP and, for larger instances, an adaptive large neighborhood search (ALNS) is proposed.

The PRP is considerably less investigated. Weidinger et al. (2018) present an ILP model and an ALNS to solve the PRP given that the sequence of pods to bring to the stations is known. Aldarondo and Bozer (2020) provide analytical formulas to determine the expected distance traveled by robots to perform a task as a function of the PSP and PRP policies, the shape of the storage area, and the locations of the stations. Li et al. (2020) consider that pods are assigned to locations using a decentralized storage policy based on a turnover rate. Simulations of an RMFS operating with this policy show a significant reduction in energy consumption and an increase in the order picking efficiency. Two studies found analyze the three problems considered here. Merschformann et al. (2019) summarize the most common policies of the literature and practice and sequentially solve all three problems in real time. The OAP is solved when an order is fulfilled, which triggers another order from the backlog to be assigned to the station. The PSP is solved when a robot working for a station can perform a new task. Finally, the PRP is solved when a pod leaves a station. For each of the problems, several solution policies are suggested. The RMFS operating with different combinations of policies is evaluated using a simulation framework to determine which one performs best considering several performance measures. The experiments show that cross-dependencies exist between the policies used. Rimélé et al. (2021) present a mathematical framework to model the decisions considering the stochastic nature of processing times and demands. The decision process is formalized for a real time picking strategy using a dynamic stochastic model. The model is illustrated using simple rules, similar to those used in Merschformann et al. (2019).

Compared to the literature analyzed, our major contributions are summarized as: (i) we extend the ILP model from Xie et al. (2021) to integrate the PRP decisions with the OAP and the PSP; (ii) we analyze the impact of integrating these problems on the energy consumption in an RMFS, since only the PRP is considered in Li et al. (2020); (iii) pod repositioning in wave picking is also new since previous studies, such as Merschformann et al. (2019) and Rimélé et al. (2021), considered a real time picking strategy only; and (iv) we present the first stochastic programming model using a sampling scheme to account for the uncertainty of future demands in wave picking. The previous solution approaches either considered deterministic versions or real-time simulations to solve these problems.

5.3 Problem description

In the RMFS considered, products are stored in identical storage pods. Robots can move underneath the pods, lift them, and carry them to where they are demanded. The storage area has a grid format, where each square represents either an aisle, used by robots to carry a pod through, or a storage location, either containing a parked pod or not. As commonly considered in the RMFS literature, both vertical and horizontal aisles are one-way and directions alternate among parallel aisles (Lamballais et al., 2017; Merschformann et al., 2019). Storage locations are grouped in blocks divided by rows, each with a storage location on each side, thus allowing pods to have direct access to aisles. Picking stations are equally distributed on one side of the storage area and a buffer zone separates the stations from the storage area. The ordered products are picked by stationary pickers from the pods carried to the stations. Figure 5.1 shows the floor plan representation of the described storage area.

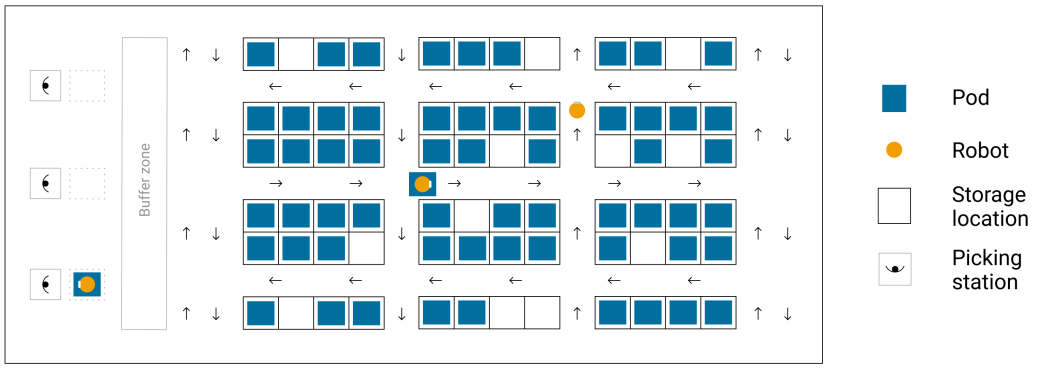


Figure 5.1 – Representation of an RMFS storage area layout

5.3.1 Decision problems

In wave picking, a planning horizon is divided into multiple periods, each representing a wave. Orders that arrived in the previous period, for example, overnight, are in a backlog and can be picked in the first wave. Each order contains a set of distinct order lines, i.e., different products to be picked. The OAP has to be solved to determine which orders will be handled by which stations (Merschformann et al., 2019). In the OAP, multiple orders are batched and assigned together to stations. Overall, larger batch sizes are preferred for energy-efficient picking. However, batch sizes are limited by the capacity of stations, determined by the maximum number of bins that picked products are deposited that are available in the station at a time. To avoid further order consolidation operations, we consider that each bin is used to deposit products from a single order, although combining order lines is also possible when orders can be split among stations (Xie et al., 2021). Another common consideration is to balance pickers' workload, such that each picker performs a similar number of picks in each wave (Valle and Beasley, 2021; Zhuang et al., 2021). To account for fairness among works

distributed to pickers, we consider workload balance in our models. Workload balance is modeled such that the difference between the number of order lines assigned in a wave to all pairs of pickers does not exceed a threshold.

In a sequential approach, given an OAP solution, the next decision concern which pods containing the demanded products should be carried to each station. Typically, scattered storage is adopted in an RMFS, such that items of the same product are spread over the warehouse in multiple pods (Weidinger and Boysen, 2018). This increases the probability of having some nearby pods carrying a requested product, reducing the mean processing time of orders (Boysen et al., 2017; Lamballais et al., 2020). Due to the scattered storage, we have to decide which pods should be carried to the stations to meet the demands of the orders assigned to them by solving the PSP. We consider that pod replenishment is done before the beginning of the planning horizon such that the inventory in each pod is known when planning the picking waves. We also assume that a sufficient quantity of items to satisfy all orders for the planning horizon are available in each pod, which is a common and reasonable assumption in practice (Zhuang et al., 2021). The initial pod locations are also known, which can be randomly generated or determined by a storage policy, such as zoning (Lamballais et al., 2017). When solving the OAP and the PSP in an integrated manner, the same products contained in different orders are usually batched together and assigned to a single station. Thus, we avoid the same pod being used by multiple stations within a wave. For this reason, we also assume that each pod is carried to a station only once per wave.

Still in the sequential approach, once the OAP and the PSP are solved, it is time to plan the robots' tasks. A task indicates where each robot should go and the path to be followed. A central server is responsible for assigning all tasks to robots. The decisions about which robots will be selected consider their positions and the arrival sequence of tasks (Gharehgozli and Zaerpour, 2020; Li and Liu, 2016). When deciding the storage location for returning a pod once the picking at a station is done, we solve the PRP (Xie et al., 2021). When a pod is returned, it can be repositioned at any available storage location, i.e., a location that has a space to park the pod. In our models to integrate the PRP with other problems, we consider that all locations left empty by the pods demanded in the current wave are available for all pods at their return. The rearrangement of pod locations is an important aspect to be considered in a dynamic context, such as wave picking. The reason is that pods containing products that will be demanded in a future wave can be positioned near picking stations, saving time and energy.

5.3.2 Energy consumption

From a context of sustainable development, the objective when solving the previously mentioned problems is to minimize the energy consumed by robots. Energy saving has been the most frequently studied topic within the context of green warehousing (Bortolini et al., 2019a).

Typical robot's tasks in an RMFS are done following these steps:

Step (i) Move the unloaded robot from its current position to a requested pod;

Step (ii) Lift the pod;

Step (iii) Move the loaded robot to the designated station;

Step (iv) Stop while picking is performed;

Step (v) Move the loaded robot to place the pod in its new position;

Step (vi) Drop the pod.

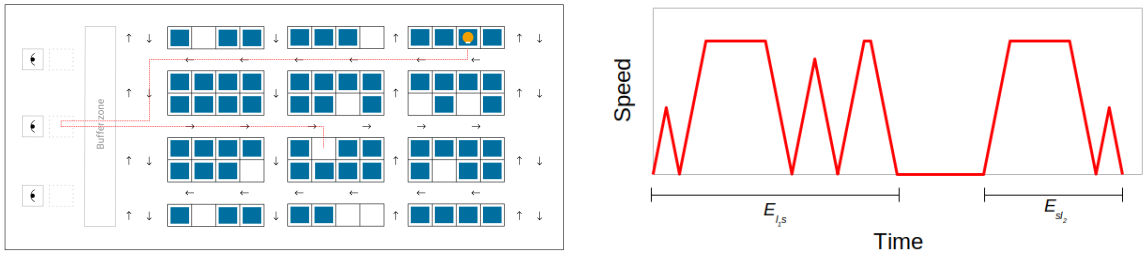
The steps described ignore certain situations, such as blocking, obstacles that may appear in the robot's path forcing them to stop, and queues formed by robots waiting to be processed by stations. Blocking and queues are not an issue in the wave picking modeled here since time constraints are not considered, so robots carrying pods can wait until the path is entirely free at no energy cost. Unexpected obstacles rarely occur during operations such that their impact is negligible. The energy consumed to move a loaded robot is 2.5 times higher than moving an unloaded one (Zou et al., 2018). Since most of the time robots are loaded, the effect on unloaded moves (i) has a low impact on the total energy consumed and is, therefore, disregarded here. For this reason, tasks that unloaded robots move to recharge stations are also not relevant for this study. So, the total energy consumed $E_{l_1 s l_2}$ in the pertinent steps of a robot task to carry a pod from a location l_1 to a station s and back to a location l_2 is

$$E_{l_1 s l_2} = E_{lift} + E_{l_1 s} + E_{s l_2} + E_{drop}, \tag{5.1}$$

where the energy consumed is E_{lift} to lift the pod at its initial storage location l_1 (ii), $E_{l_1 s}$ to move from l_1 to a station s (iii), $E_{s l_2}$ to return the pod to a storage location l_2 (v), and E_{drop} to drop the pod in l_2 (vi). The energy spent in (iv) is negligible.

Robots paths are computed as the shortest path between storage locations and stations following the grid layout of the storage area and the directions of the aisles. To move straight between two points, a robot accelerates, reaches the maximum speed and keeps moving if the path is long enough, and then decelerates. If the path is too short, the robot does not reach its maximum speed, so the energy consumed is only a fraction of the energy spent during the acceleration/deceleration. The total energy spent in a full path between a storage location and a station, or vice-versa, is then the sum of the energy spent to perform all the straight paths contained in it. Figure 5.2 is based on Li et al. (2020) and shows an example of the speed changes in a path traveled by a robot. Paths are short in the example when moving the pod from its storage location to the aisle, and vice-versa, and when entering in front of the station. Note in Figure 5.2b that speed changes are split into a section representing the path

traveled by the robot when carrying the pod to the station ($E_{l_1 s}$), then it sits there while the picks are performed, and finally, it returns the pod to its new storage location (E_{sl_2}).



(a) Path traveled

(b) Speed change and energy consumed

Figure 5.2 – Path and speed of a robot to perform a task

The energy consumed by a robot traveling at a constant speed is

$$E_c = e_u \frac{d_c}{v_{max}}, \tag{5.2}$$

where e_u denotes the energy consumed by a unit of time, d_c is the distance traveled at a constant speed, and v_{max} is the maximum speed of a robot. The energy consumed by a robot during acceleration and deceleration is

$$E_a = E_d = e_u \frac{v_{max}}{a}, \tag{5.3}$$

where a is the acceleration/deceleration of a robot. In summary, $E_{l_1 s}$ and E_{sl_2} are the sum of all straight paths performed by a robot, and each straight path is the sum of E_a, E_c, and E_d, depending on its length.

Zou et al. (2018) estimate the work to lift/drop a pod as $E_{lift} = E_{drop} = 0.8$ kJ. Li et al. (2020) estimate that a robot charged with 2.4 kWh of energy will operate for about 6 h. Therefore, the average hourly energy consumption rate of this robot is $e_u = 0.4$ kWh or 0.4 kJ/s. They also assume a maximum speed of $v_{max} = 2$ m/s and an acceleration of $a = 1$ m/s^2. We consider that the distance between the center of each square on the square grid layout is equal to one meter. With the given equations and parameters, we can estimate the total energy consumed by a robot to perform a task.

5.4 Mathematical models for the integrated OAP–PSP–PRP

Next, we present the mathematical formulations to solve the OAP, PSP, and PRP in an integrated manner for the RMFS previously described. For ease of reference, Table 5.2 provides a summary of the notation used to model the integrated problem. Other notation used throughout this section will be introduced when needed.

Table 5.2 – Notation for the integrated OAP–PSP–PRP dynamic model

Sets	
$\mathcal{L}$	Storage locations
$\mathcal{S}$	Picking stations
$\mathcal{P}$	Pods
$\mathcal{I}$	Products
$\mathcal{P}_i \subseteq \mathcal{P}$	Pods that contain product $i \in \mathcal{I}$
$\mathcal{W} = \{1, \ldots, \lvert\mathcal{W}\rvert\}$	Waves
$\mathcal{O}$	Orders
$\mathcal{O}_w \subseteq \mathcal{O}$	Orders arrived in wave $w \in \mathcal{W}$
$\mathcal{I}_o \subseteq \mathcal{I}$	Order lines (products) of an order $o \in \mathcal{O}$
Parameters	
$E_{l_1 s l_2}$	Energy consumed by a robot to carry a pod from location l_1 to station s and return to location l_2
L_p	Initial location of pod p
C	Stations capacity (in orders) in a wave
δ	Maximum difference of order lines assigned to pickers in a wave
Decision variables	
x_{ps}^w	(PSP) Whether pod p is assigned to station s in wave w
y_{os}^w	(OAP) Whether order o is assigned to station s in wave w
y_{ios}^w	Whether product i of order o is assigned to station s in wave w
z_{pl}^w	(PRP) Whether pod p is parked at location l at the end of wave w

Regardless of the strategy used to solve the integrated problem, we are given a set of storage locations $\mathcal{L}$, picking stations $\mathcal{S}$, pods $\mathcal{P}$, and products $\mathcal{I}$. The set $\mathcal{P}_i \subseteq \mathcal{P}$ represents all pods that contain a product i. The initial location of a pod p is given by L_p. The maximum number of orders a station can handle in a wave is given by C. A parameter δ is used to define the maximum difference in the number of picks (order lines) performed by all pairs of pickers. Setting $\delta = 0$ will impose the same number of picks to be performed at each station. However, setting a larger δ allows more flexibility when distributing picking tasks. As discussed in Section 5.3.2, the energy consumed by a robot to carry a pod from a storage location l_1 to a station s and return to a location l_2 is represented by $E_{l_1 s l_2}$.

Since we consider wave picking, all decisions are made dynamically at the beginning of each wave. So, we are given $\mathcal{W} = \{1, \ldots, |\mathcal{W}|\}$ as the set of waves to be planned. Let $\mathcal{O}$ represent the set of all orders that will arrive in the planning horizon each containing the order lines (products) $\mathcal{I}_o \subseteq \mathcal{I}$. The set $\mathcal{O}_w \subseteq \mathcal{O}$ indicates all orders added to the backlog at the beginning of the wave w.

Three types of decision variables are considered, each representing one of the problems being integrated. Binary variables x_{ps}^w represent the PSP decisions and indicate whether pod p is assigned to station s in wave w. Binary variables y_{os}^w represent the OAP decisions and indicate whether order o is assigned to station s in wave w. It is also required to define

variables y^w_{ios} indicating whether product i demanded by order o is assigned to station s in wave w. Finally, binary variables z^w_{pl} represent the PRP decisions and indicate whether pod p is parked at location l at the end of wave w, regardless of this pod being used in this wave. For convenience, we consider z^0_{pl} as the binary equivalent of the parameter L_p to represent whether pod p is initially located in l.

5.4.1 A dynamic integer non-linear programming model for the case with known demands

The OAP, PSP, and PRP for the RMFS presented are integrated using a DINLP model to consider the decisions to be made in all waves. This model works as an oracle and assumes that all orders that will arrive in each wave are known *a priori*. Although this is a strong assumption for a real-world application in e-commerce distribution centers, we will show later that this model can be adapted to be used with the data available for demands with uncertainty. The integrated problem is modeled as follows.

$$\min \sum_{w\in\mathcal{W}}\sum_{p\in\mathcal{P}}\sum_{s\in\mathcal{S}}\sum_{l_1\in\mathcal{L}}\sum_{l_2\in\mathcal{L}} E_{l_1 s l_2} z^{(w-1)}_{pl_1} x^w_{ps} z^w_{pl_2} \tag{5.4}$$

subject to

$$\sum_{o\in\mathcal{O}_w}\sum_{s\in\mathcal{S}} y^w_{os} = |\mathcal{O}_w|, \quad \forall w \in \mathcal{W} \tag{5.5}$$

$$\sum_{s\in\mathcal{S}} y^w_{os} = 1, \quad \forall w \in \mathcal{W}, o \in \mathcal{O}_w \tag{5.6}$$

$$y^w_{os} = y^w_{ios}, \quad \forall s \in \mathcal{S}, w \in \mathcal{W}, o \in \mathcal{O}_w, i \in \mathcal{I}_o \tag{5.7}$$

$$\sum_{p\in\mathcal{P}_i} x^w_{ps} \geq y^w_{ios}, \quad \forall s \in \mathcal{S}, w \in \mathcal{W}, o \in \mathcal{O}_w, i \in \mathcal{I}_o \tag{5.8}$$

$$\sum_{s\in\mathcal{S}} x^w_{ps} \leq 1, \quad \forall p \in \mathcal{P}, w \in \mathcal{W} \tag{5.9}$$

$$\sum_{o\in\mathcal{O}_w} y^w_{os} \leq C, \quad \forall s \in \mathcal{S}, w \in \mathcal{W} \tag{5.10}$$

$$\sum_{o\in\mathcal{O}_w}\sum_{i\in\mathcal{I}_o} \left|y^w_{ios_1} - y^w_{ios_2}\right| \leq \delta, \quad \forall s_1, s_2 \in \mathcal{S}, w \in \mathcal{W} \tag{5.11}$$

$$\sum_{p\in\mathcal{P}} z^w_{pl} \leq 1, \quad \forall l \in \mathcal{L}, w \in \mathcal{W} \tag{5.12}$$

$$\sum_{l\in\mathcal{L}} z^w_{pl} = 1, \quad \forall p \in \mathcal{P}, w \in \mathcal{W} \tag{5.13}$$

$$\left|z^{(w-1)}_{pl} - z^w_{pl}\right| \leq \sum_{s\in\mathcal{S}} x^w_{ps}, \quad \forall p \in \mathcal{P}, l \in \mathcal{L}, w \in \mathcal{W} \tag{5.14}$$

$$x^w_{ps} \in \{0,1\}, \quad \forall p \in \mathcal{P}, s \in \mathcal{S}, w \in \mathcal{W} \tag{5.15}$$

$$y^w_{os}, y^w_{ios} \in \{0,1\}, \quad \forall s \in \mathcal{S}, w \in \mathcal{W}, o \in \mathcal{O}_w, i \in \mathcal{I}_o \tag{5.16}$$

$$z_{pl}^{w} \in \{0,1\}, \quad \forall p \in \mathcal{P}, l \in \mathcal{L}, w \in \mathcal{W}. \tag{5.17}$$

The objective function (5.4) minimizes the energy consumed by robots to perform all tasks assigned to them during the planning horizon. The cubic expression indicates a task performed by a robot, meaning that a pod initially located in l_1 as defined in a previous wave ($z_{pl_1}^{(w-1)} = 1$) is carried in this wave to station s ($x_{ps}^{w} = 1$) and is returned to location l_2 ($z_{pl_2}^{w} = 1$). Then, the energy cost to perform this task is $E_{l_1 s l_2}$.

Constraints (5.5) set the number of orders satisfied in a wave equal to the number of orders arrived in that wave. Constraints (5.6) guarantee that orders are assigned to a single station. Constraints (5.7) ensure that if an order is assigned to a station in a wave, then all its order lines are also assigned to the same station in the same wave. Constraints (5.8) assure that at least one pod containing products assigned to a station will be carried to it when required. Constraints (5.9) determine that each pod can only visit one station in each wave. Capacity constraints (5.10) guarantee that the number of orders assigned to the stations respects their capacities. The workload balance is guaranteed by constraints (5.11) by imposing that the difference in the number of picks performed in each pair of stations s_1 and s_2 is lower than threshold δ. Constraints (5.12)–(5.14) are used to solve the PRP by ensuring that pods are assigned to valid locations after each wave. Constraints (5.12) guarantee that no more than one pod is parked at any location after the wave. Constraints (5.13) guarantee that all pods are assigned to a single location at any time. Finally, constraints (5.14) impose that pods stay in the same location ($|z_{pl}^{(w-1)} - z_{pl}^{w}| = 0$) when not moved in a wave ($\sum_{s \in \mathcal{S}} x_{ps}^{w} = 0$). We highlight that the opposite is not always true since a pod can move and still return to the same location. The domain of the decision variables is defined in constraints (5.15)–(5.17).

Linearization

The MIP model presented is non-linear due to the cubic term in the objective function and the module function in the constraints (5.11) and (5.14). To make it solvable by a commercial solver for linear programming, we can linearize the former by replacing the product abc of three binary variables a, b, and c, by a new binary variable d, adding the constraints $d \leq a$, $d \leq b$, $d \leq c$, and $d \geq a + b + c - 2$ to the model. Meanwhile, the latter is linearized by replacing the constraint in the form of $|a| \leq b$ by two new constraints $a \leq b$ and $a \leq -b$.

Valid inequality

A valid inequality for the DINLP model described consists of removing the furthest locations from the decision variables when a pod has to return from a station. The rationale is that it will never be optimal to return a pod after a pick to a storage location beyond the $|\mathcal{P}|$ closest locations from the station it was assigned to, i.e., to park in the furthest $|\mathcal{L}| - |\mathcal{P}|$ locations from this station. Since distances are station-dependent, it is not possible to simply remove a

set of locations from the model. Instead, the locations are sorted by distance to each station and, then, the inequality

$$x_{ps}^{w} + z_{pl}^{w} \leq 1, \quad \forall p \in \mathcal{P}, l \in \mathcal{L}_p, s \in \mathcal{S} \tag{5.18}$$

is added to the model to guarantee that when a pod p is assigned to a station s, it cannot return to a location l in the set $\mathcal{L}_p$ containing the $|\mathcal{L}| - |\mathcal{P}|$ furthest locations from p.

Lower bound

A lower bound to the problem can be input to the solver to speed up the optimization process. For the problem considered, it can be retrieved from the minimum number of pod visits to satisfy the demands in each picking wave. To calculate a lower bound for the minimum number of pod visits in a wave, we first get the number of distinct products demanded in this wave (U). Then, we identify the pod with the maximum number of products and the number of demanded products it contains (V). Knowing U and V, a lower bound for the number of pod visits is given by $W = \lceil U/V \rceil$. An example to illustrate is as follows. If among the orders in the backlog we have to pick nine distinct products and the pod containing the most products among them has four products, we can be assured that no less than $W = \lceil 9/4 \rceil = 3$ pods are required to meet all demands in this wave. Since our model minimizes energy consumption, not pod visits, we have to transform the lower bound described in terms of energy.

Given the minimum number of pod visits, a lower bound for the energy consumption is computed in a two-phase process. In the first phase, we have to find the W nearest pods containing at least one of the products demanded in this wave, which is a lower bound for the PSP. In the second-phase, we have to return them to the nearest available locations, which is a lower bound for the PRP. The detailed steps for this procedure are as follows:

Step (i) Sort all pods by their distances to the nearest station;

Step (ii) Get the W nearest pods in the sorted list that contain at least one product demanded in this wave;

Step (iii) Add to the lower bound the distances of these pods to their nearest stations;

Step (iv) Sort all available locations, including those left empty by the W nearest pods, by the nearest distance to each station;

Step (v) Given the nearest stations of the W nearest pods chosen, add to the lower bound of the distances to return them to the nearest available locations.

Model reduction

Another significant improvement is possible by reducing the number of locations $|\mathcal{L}|$ in half since the energy cost to bring a pod from any row is similar, regardless of the side the pod is parked. In this case, it is enough the reformulate constraints (5.12) to

$$\sum_{p \in \mathcal{P}} z_{pl}^{w} \leq 2, \quad \forall l \in \mathcal{L}, w \in \mathcal{W}, \tag{5.19}$$

and apply it on the reduced set $\mathcal{L}$.

5.4.2 Solution approaches for the case with demand uncertainty

Warehouses may opt for wave picking to simplify the decision-making on the order picking process so that the operational decisions can be made periodically instead of in real-time. Wave picking also has the advantage of creating order batches more efficiently. The larger pool of orders in the backlog allows different products located in the same pod to be batched and picked together. This results in a reduction in pod visits, as shown by many previous studies, but may decrease the energy consumed by robots when pods parked in better locations are used for the picks.

In the optimization context, wave picking can be seen as a sequential decision problem in which decisions are made in an iterative process between "decide" and "reveal new information". This strategy belongs to the *a priori optimization* modeling paradigm since the OAP, PSP, and PRP decisions are made given the current state of the warehouse considering that the uncertainty may affect the outcome. The main source of uncertainty in the RMFS lies in the future products to be picked. Therefore, in practice, warehouse managers may opt for approaches that account for the demand uncertainty to solve the integrated problem. Among the factors to be considered are the possibility of forecasting demands and the degree of difficulty to adopt the approach chosen.

The following sections describe the solution approaches considered in this study. In the *sequential* approach, we consider the most common method used in the literature where the problem is decomposed into two subproblems solved sequentially, i.e., first, the OAP and the PSP are solved together, then the PRP alone. In the *myopic* approach, we assume that no information about future demands is available or that the problem is solved with no lookahead. Therefore, each wave must be planned considering only the current state of the system. Finally, in the *stochastic* approach, we assume that the information about future demands is known at a stochastic level when planning for a wave. This information can be embedded in the model, helping find robust solutions that are expected to lead to good solutions for the next wave by explicitly considering the expected future costs associated with the decisions made at the current wave.

The mathematical models for these approaches are derived from the DINLP model presented before. We highlight that using any of these approaches to solve the integrated problem will necessarily result in a solution worse than the optimal solution found solving the DINLP model if the predicted scenario used to define the DINLP turns out to be exactly observed. For this reason, we will refer henceforth to the integrated problem with known demands as the *optimal* (oracle) approach. Table 5.3 summarizes the approaches considered, showing what tasks they are optimizing and the type of objective function considered.

Table 5.3 – Summary of the solution approaches considered for the integrated OAP–PSP–PRP

Approach	*Optimality*	*Objective function*
Sequential	Bring pods in the 1^{st} wave only	Linear
Myopic	1^{st} wave only	Quadratic
Stochastic	1^{st} wave + expected future cost	Stochastic
Optimal (oracle)	All waves	Cubic

Sequential approach

In the sequential approach, the integrated problem is solved in a two-phase process for each wave using the information available for the orders currently in the backlog, i.e., the set of orders $\mathcal{O}_w$ that arrived in this wave w. The previous DINLP model is decomposed into two subproblems which are solved sequentially. The first subproblem is the integrated OAP–PSP, while the second one is the PRP alone. This approach allows us to verify the impact of not integrating pod repositioning with the other problems. The integrated OAP–PSP can be adapted from the DINLP model by removing PRP decisions. The dynamic model is reformulated to remove the index w from all decision variables. The result is an ILP formulated as:

$$\min \sum_{l \in \mathcal{L}} \sum_{s \in \mathcal{S}} (E_{lift} + E_{L_{ps}}) x_{ps} \tag{5.20}$$

subject to

$$\sum_{o \in \mathcal{O}_w} \sum_{s \in \mathcal{S}} y_{os} = |\mathcal{O}_w|, \tag{5.21}$$

$$\sum_{s \in \mathcal{S}} y_{os} = 1, \quad \forall o \in \mathcal{O}_w \tag{5.22}$$

$$y_{os} = y_{ios}, \quad \forall s \in \mathcal{S}, o \in \mathcal{O}_w, i \in \mathcal{I}_o \tag{5.23}$$

$$\sum_{p \in \mathcal{P}_i} x_{ps} \geq y_{ios}, \quad \forall s \in \mathcal{S}, o \in \mathcal{O}_w, i \in \mathcal{I}_o \tag{5.24}$$

$$\sum_{s \in \mathcal{S}} x_{ps} \leq 1, \quad \forall p \in \mathcal{P} \tag{5.25}$$

$$\sum_{o \in \mathcal{O}_w} y_{os} \leq C, \quad \forall s \in \mathcal{S} \tag{5.26}$$

$$\sum_{o \in \mathcal{O}_w} \sum_{i \in \mathcal{I}_o} |y_{ios_1} - y_{ios_2}| \leq \delta, \quad \forall s_1, s_2 \in \mathcal{S} \tag{5.27}$$

$$x_{ps}, y_{os}, y_{ios} \in \{0, 1\}, \quad \forall p \in \mathcal{P}, s \in \mathcal{S}, o \in \mathcal{O}_w, i \in \mathcal{I}. \tag{5.28}$$

The objective function (5.20) represents the total energy consumed by robots to bring the pods demanded from their storage locations to the stations. Constraints (5.21)–(5.27) are equivalent to constraints (5.5)–(5.11) for the OAP and the PSP decisions, and constraints (5.28) define the domain of the variables of the OAP and the PSP. The resulting linear model is an adaptation of the ILP model presented in Xie et al. (2021) to consider energy consumption as the system performance measure and a workload balance to ensure a fair amount of work distributed among pickers.

This ILP model optimizes the energy consumption to bring pods to stations. However, it is still required to return pods to storage locations to complete all decisions for a wave. Since we do not consider the order of pods arriving at stations, this PRP is a simplification of the problem presented in Weidinger et al. (2018). Given the set of pods located at picking stations, defined by solving the previous OAP and PSP model, and the set of storage locations available in the storage area, a simple way to approximate the optimal solution for this PRP is by using the nearest rule presented in Merschformann et al. (2019), where each pod is returned to the nearest available storage location to the station they are in. This rule is effective since parking pods further than the nearest available location is undesirable since it will result in a higher energy cost. We still note that any combination of assignments of pods in a station to the nearest available storage locations results in the same PRP solution. Despite its simplicity, Merschformann et al. (2019) show that the nearest rule still performed best in most cases. In the dynamic context analyzed, however, it may lead to bad solutions in the long term since it can park pods that will not be required for a long time in good locations.

Myopic approach

The myopic approach can be seen as the integrated version of the sequential approach since all three problems are solved using a single framework. Following the notation previously introduced, the myopic approach is modeled as follows.

$$\min \sum_{p \in \mathcal{P}} \sum_{s \in \mathcal{S}} \sum_{l \in \mathcal{L}} E_{L_p s l} x_{ps} z_{pl} \tag{5.29}$$

subject to (5.21)–(5.28) and to

$$\sum_{p \in \mathcal{P}} z_{pl} \leq 1, \quad \forall l \in \mathcal{L} \tag{5.30}$$

$$\sum_{l \in \mathcal{L}} z_{pl} = 1, \quad \forall p \in \mathcal{P} \tag{5.31}$$

$$|1 - z_{pl}| \leq \sum_{s \in \mathcal{S}} x_{ps}, \quad \forall p \in \mathcal{P}, l \in \mathcal{L} \tag{5.32}$$

$$z_{pl} \in \{0,1\}, \quad \forall p \in \mathcal{P}, l \in \mathcal{L}. \tag{5.33}$$

As in the sequential approach, the objective function and all constraints are adapted from the dynamic model to consider that decisions are made for a single wave. So, the objective function (5.29) considers the tasks performed to move a pod p from its initial location L_p to the station it is required and back to any available space in the storage area. The myopic approach considers all constraints from the sequential approach, setting the new constraints (5.30)–(5.32) for the PRP decisions. Compared to the DINLP model for the optimal approach, this model has two major implications. First, its size is significantly reduced without the additional index for the waves. Also, the objective function of the model is now quadratic instead of cubic, which can be linearized by replacing the quadratic term ab by a new variable c, adding the constraints $c \leq a$, $c \leq b$, and $c \geq a + b - 1$ to the model. The valid inequalities, lower bound, and model reduction described in Section 5.4.1 are easily adapted for the myopic model considering a single wave scenario.

Stochastic approach

Given the current state of the RMFS, we have shown that the myopic approach can be used to make successive decisions for the dynamic integrated problem. However, since it only considers short-term costs, its solution can place pods with a low turnover in good positions in the storage area, reducing the efficiency of picking in future waves. An alternative way to solve the dynamic problem is by integrating the expected behavior of future demands arriving in the next wave to the model using stochastic programming, resulting in the so-called stochastic approach. Predicting future demands is a challenge for many warehouses using the RMFS. However, advances in tools for regression analysis, such as neural networks, are improving predictions for the short and medium terms classically made using time series estimators and other machine learning methods (Garnier, 2021).

Consider an arbitrary solution $u^w = \{x^w, y^w, z^w\}$ for the integrated problem in a picking wave w. The expected cost of u^w is given by $f(u^w, z^{w-1}) + \mathbb{E}[w+1]$, where $f(u^w, z^{w-1})$ is the cost of u^w for the initial pod locations z^{w-1}, and $\mathbb{E}[w+1]$ is the expected cost of the subsequent wave $w+1$. Assuming continuous distributions for the uncertain demands for the wave $w+1$, we can approximate $\mathbb{E}[w+1]$ by sampling a set Ω of scenarios to represent the backlog state at the beginning of $w+1$, where each scenario $\xi \in \Omega$ sampled has a probability of occurrence $P(\xi)$. The cost of an arbitrary solution u_ξ^{w+1} for the scenario ξ with the pods initially arranged as defined by u^w is given by $f(u_\xi^{w+1}, z^w)$. We ignore the expected cost of the next wave, $\mathbb{E}[w+2]$ for the sake of avoiding the *curse of dimensionality* to allow some optimization potential. Therefore, the expected cost of wave $w+1$ for the arbitrary solutions for each backlog sampled is $\mathbb{E}[w+1] = \sum_{\xi \in \Omega} P(\xi) f(u_\xi^{w+1}, z^w)$. In the stochastic approach, we are interested in finding the optimal solution u^w amongst the solutions space U^w representing all feasible solutions for the integrated problem in a given wave w. We do that by solving a

two-stage stochastic programming model where the first stage is defined as

$$\min_{u^w \in U^w} f(u^w, z^{w-1}) + \mathbb{E}_{\xi \in \Omega}[Q(u^w, \xi)], \tag{5.34}$$

where $Q(u^w, \xi)$ is the optimal solution for the subproblem

$$\min_{u^{(w+1)} \in U^{(w+1)}} f(u_\xi^{(w+1)}, z^w), \tag{5.35}$$

solved in the second stage for a given scenario $\xi \in \Omega$.

The myopic model presented in Section 5.4.2 is adapted to solve this two-stage stochastic programming model using a logic-based Benders decomposition technique (Hooker and Ottosson, 2003). In the first stage, it is enough to modify the objective function (5.29) for the myopic approach to

$$\min \sum_{p \in \mathcal{P}} \sum_{s \in \mathcal{S}} \sum_{l \in \mathcal{L}} E_{L_p sl} x_{ps}^w z_{pl}^w + \sum_{\xi \in \Omega} P(\xi) f(z_{pl}^w, \xi), \tag{5.36}$$

where the term $P(\xi)f(z_{pl}^w, \xi)$ is added to represent the stochastic costs to be estimated. To this end, we solve a similar model in the second-stage to find the optimal solution for the set of orders $\mathcal{O}^\xi$ that are expected to arrive in each scenario $\xi \in \Omega$. A scenario ξ is solved for the objective function

$$\min f(z_{pl}^w, \xi) = \sum_{p \in \mathcal{P}} \sum_{s \in \mathcal{S}} \sum_{l \in \mathcal{L}} E_{(lz_{pl}^w)sl} x_{ps}^{w+1} z_{pl}^{w+1}, \tag{5.37}$$

where lz_{pl}^w represents the index of the initial location of a pod p in this stage given by the solution found in the first stage, and the second-stage variables x^{w+1}, y^{w+1}, and z^{w+1} are subject to the myopic constraints. The sampling technique used to generate $\mathcal{O}^\xi$ is presented in Section 5.5.2 and the samples are generated as described in Section 5.6.2.

In the optimization process implemented, we replace the stochastic cost term $P(\xi)f(z_{pl}^w, \xi)$ in the objective function by a new decision variable c. Whenever a new feasible solution for the first-stage model is found we solve the second-stage model for each scenario $\xi \in \Omega$ to find $f(z_{pl}^w, \xi)$. Then, we add a cut to the first-stage model as

$$c \geq \left(1 - \sum_{p \in \mathcal{P}} \sum_{l \in \mathcal{L}} (1 - z_{pl}^w)\right) \sum_{\xi \in \Omega} P(\xi) f(z_{pl}^w, \xi), \tag{5.38}$$

indicating that c should be at least equal to the expected cost $\sum_{\xi \in \Omega} P(\xi) f(z_{pl}^w, \xi)$ whenever all variables z_{pl}^w values reappear in the optimization process.

It is possible to improve the optimization process described from a lower bound derived for the stochastic costs. Let $LB(\xi)$ be a known lower bound for $f(z_{pl}^w, \xi)$. The lower bound for the stochastic costs in (5.36) is $\sum_{\xi \in \Omega} P(\xi) LB(\xi)$. Adding this lower bound to (5.36) can significantly speed up the solving process since the second-stage model will be solved only when the stochastic costs can improve the current solution given the lower bound provided.

Given the improvement described, we still have to subtract $LB(\xi)$ from the objective function of the second-stage model for scenario ξ to compensate the stochastic costs already considered in the first stage. In our experiments, the lower bound $LB(\xi)$ is calculated using a modified version of the method described in Section 5.4.1 since we do not know the initial pod locations for the second stage before solving the first-stage model. Now, we consider that the minimum number of pods W to be picked will have to traverse the buffer zone twice without identifying their possible initial locations.

5.5 A local search matheuristic for the integrated OAP–PSP–PRP with demand uncertainty

Due to the complexity of solving most of the models presented, the conception of a heuristic is required to approximate the optimal solution for the integrated problem with stochastic demands for large-size instances. We design a local search matheuristic. This matheuristic combines the effectiveness of generating solutions using one of the mathematical models previously introduced with a local search capable of quickly verifying a neighborhood based on the repositioning of pods.

The matheuristic is divided into three parts detailed in the next sections. The first part generates a feasible solution for the problem from a simple adaptation of the sequential approach. Then, we explain how we can generate representative scenarios to estimate the stochastic cost of this solution. Finally, a very efficient local search is described that can improve the initial solution by searching in a neighborhood defined by solutions generated by swapping pod positions in the current wave. The structure of our search allows the stochastic cost to be easily updated for each neighbor solution using simple analytical formulas.

5.5.1 Generating feasible solutions

A feasible solution for a wave requires determining which orders are assigned to which stations, which pods are assigned to which stations, and where each pod should return after the picks, considering the constraints previously described, such as the picking stations' capacities and workload balance. The first two decisions pose the biggest challenge to generating a feasible solution since the number of existing matches between products contained in orders and pods is huge when scattered storage is used. Previous studies attempted to design heuristics for the integrated OAP–PSP (Jiang et al., 2020; Valle and Beasley, 2021; Xiang et al., 2018; Xie et al., 2021). A common technique is to use mathematical models, either entirely or partially, to obtain a feasible solution that a local search algorithm can later improve.

In this study, due to the reduced complexity of the ILP model presented for the integrated OAP–PSP compared to the other models presented, we use this model to generate a set of feasible solutions for the integrated OAP–PSP. State-of-the-art mathematical programming

solvers can generate a pool of solutions for a problem that are certified to be the N best solutions for it. So, we start our matheuristic by solving the ILP model until N solutions are proven to be the best possible ones. Then, the PRP is solved using the nearest rule to decide where the pods should be returned to after the picks. All solutions are evaluated and only the best one is kept.

In preliminary experiments, we observed that a too high N value does not lead to good solutions since an improving one for the OAP–PSP–PRP is rarely too far from the optimal one for the OAP–PSP. Therefore, we fixed N to 100 in all our experiments. Since this heuristic starts from the optimal solution of the sequential approach and possibly improves it for the current wave, it leads to an energy cost that is upper bounded by the optimal solution for the sequential approach and lower bounded by the solution for the myopic approach. We show by our computational experiments (Section 5.6) that, despite its simplicity, our matheuristic improves the initial solutions to be closer to those given by the myopic approach.

5.5.2 Evaluating the stochastic cost

Given a feasible solution for the current wave, we estimate the stochastic cost of the next wave by sampling possible scenarios that may be observed in the future. We sample a number of scenarios S using a sample average approximation (SAA) scheme. The SAA is used to solve stochastic problems using a Monte Carlo simulation. It considers that a random sample of scenarios drawn from known probability distribution functions can approximate well the expected cost of all possible scenarios (Kleywegt et al., 2002). We use the SAA to sample a certain number of scenarios using typical demand distributions as described in Section 5.6.2, and the same scenarios sampled are reused during the whole solving process to speed up the run.

After generating the backlog of S scenarios, we evaluate the stochastic cost by considering the pods' positions determined in the heuristic to generate feasible solutions as the initial layout. Then, we solve the ILP model for the OAP–PSP for each scenario individually followed by the PRP with the nearest rule. The stochastic cost is given by multiplying the solutions found by the probability of occurrence of each scenario.

Another important remark about the evaluation of the stochastic cost using the SAA is that scenarios are evaluated independently from each other. This provides a great opportunity to perform evaluations using parallel computing. Parallelism assigns different tasks of the algorithm to different threads of a computer to speed them up, potentially linearly reducing the running time of the block of tasks being parallelized. We used parallelism when solving the OAP–PSP for each scenario, which is by far the most time-consuming task done in our matheuristic.

5.5.3 Improving the current solution

Thus far, we described how a feasible solution for the current wave of the integrated OAP–PSP–PRP is generated by our matheuristic and how its stochastic cost for the subsequent wave is estimated using the SAA technique. With this information in hand, we can improve the current solution to reduce its first and second wave costs using a simple best improvement local search algorithm based on rearranging pods after they return from stations in both waves. Algorithm 5 presents the pseudocode for the local search implemented. The idea of the best improvement search is to keep updating the solution to the best one found in its neighborhood until no more improving solution can be found (lines 1 and 33). A neighborhood of a solution is defined as all the solutions that can be generated by changing the final position of a pod moved in the current wave. Given a pod p_1 moved in this wave, it can be repositioned to any available storage location, and the difference in the energy cost of this wave is $E_{s_1 l^*} - E_{s_1 l_1}$, i.e., the cost of returning this pod from the assigned station s_1 to an available location l^* instead of returning to its current location l_1 (lines 4–7). The next step is to evaluate how this movement impacts the energy cost of the future waves represented by the sampled scenarios. For each scenario, the local search updates the initial location of p_1 and its cost in case p_1 is also moved in the wave representing this scenario (lines 12–14) and searches for the pod p_2 that when repositioned to the location left empty by p_1 after the current wave leads to the lowest difference in the energy cost Δ_2^* (lines 15–21). The savings (or increase) in the total solution cost is given by the sum of the savings in both stages (line 23). If this difference is negative, then the movement of p_1 to l^* and of p_2 to l_1 results in an improving solution to the problem (lines 25–29). Since we use the best improvement strategy, we search for all solutions in the neighborhood before deciding where the solution should be moved to (line 32). The final solution for the search described is expected to approximate well the optimal solution for the stochastic approach. In the next section, we compare its results with those found by each of the previously described approaches.

5.6 Computational experiments

In this section, we report and analyze the results of extensive computational experiments performed using the methods presented. Additionally, we compare solutions for a different objective function, i.e., minimizing the number of pod visits instead of the energy consumption, and we provide a new solution approach where we can wait for more orders to arrive before starting to plan the picking waves.

The computational environment used to run the experiments is equipped with an Intel Gold 6148 Skylake CPU with a 2.4 GHz clock. Runs were limited to use a maximum of 8 GB of RAM and four cores. All methods were implemented in C++, and the parallelism was implemented using OpenMP. The exact models were solved using Gurobi 9.5.

Algorithm 5 Best improvement local search

repeat
  Best pod repositioning for the current wave and each scenario i: $z_1^*, z_2^* \leftarrow \emptyset$;
  Best improvement: $\Delta^* = 0$;
  for all pods p_1 moved in this wave to a station s_1 and returned to a location l_1 **do**
    for all locations available l^* after the current wave **do**
      Move p_1 from l_1 to l^* at the end of the current wave;
      $\Delta_1 = E_{s_1 l^*} - E_{s_1 l_1}$; {`Update current wave return cost`}
      for all scenarios $i = \{1, \ldots, S\}$ **do**
        Δ_2^* ={Large number}
        for all pods p_2 moved in this wave to a station s_2 and returned to a location l_2 in a scenario i **do**
          $\Delta_2 = 0$;
          if p_1 is also moved in i from a location l' to a station s' **then**
            $\Delta_2 = \Delta_2 + (E_{l^* s'} - E_{l' s'})$; {`Update future wave depart cost`}
          end if
          if l_1 is available at the end of scenario i **then**
            Move p_2 from l_2 to l_1 at the end of the scenario i;
            $\Delta_2 = \Delta_2 + (E_{s_2 l_1} - E_{s_2 l_2})$; {`Update future wave return cost`}
          end if
          if $\Delta_2 < \Delta_2^*$ **then**
            $\Delta_2^* = \Delta_2$;
          end if
        end for
        $\Delta_1 = \Delta_1 + \Delta_2^*$;
      end for
      if $\Delta_1 < \Delta^*$ **then**
        $z_1^* \leftarrow \{p_1, l^*\}$;
        $z_2^* \leftarrow \{p_2, l_1\}$;
        $\Delta^* = \Delta_1$;
      end if
    end for
  end for
  Reposition pods saved in z^* to their new locations;
until No improvement is possible ($\Delta^* \geq 0$)

5.6.1 Instance generation

As is common in the warehousing literature, synthetic instances were generated to test the optimization models and the matheuristic presented. Their parameters are based on previous studies on the RMFS, most simulating real conditions found in warehouses. A summary of the parameters of the instances generated is presented in Table 5.4.

Table 5.4 – Summary of the instances generated

Layout	$\lvert\mathcal{L}\rvert$	$\lvert\mathcal{S}\rvert$	$\lvert\mathcal{P}\rvert$	$\lvert\mathcal{I}\rvert$	$\lvert\mathcal{I}_p\rvert$	$\lvert\mathcal{W}\rvert$	$\lvert\mathcal{O}_w\rvert$	$\lvert\mathcal{I}_o\rvert$	L_p	C	δ	*skewness*
Tiny	16	2	13	10	3	2	5	[1,4]	Rand	3	4	High
Small	72	2	61	20, 40	5, 10		10			6		Medium
Medium	200	3	170	50, 100	7, 15		25			10		Low
Large	504	4	428	200, 500	10, 25		50			15		

We generated instances in four different layout sizes – tiny, small, medium, large – each represented by different numbers of vertical aisles, horizontal aisles, and rows in each block.

The tiny layout is a $2 \times 1 \times 2$, meaning that it has two vertical aisles, one horizontal aisle (front and back aisles excluded), and two rows of locations in each of its four blocks. Since each row has two storage locations (one on each side), these instances have $|\mathcal{L}| = 16$ locations where pods can be parked. Small, medium and large sizes are, respectively, $3 \times 2 \times 4$, $5 \times 4 \times 4$, and $9 \times 6 \times 6$, meaning they have 72, 200, and 504 storage locations, respectively. The largest layout generated is similar to the one used in (Xie et al., 2021) for their experiments to solve the OAP–PSP. The energy cost for a task $E_{l_1 s l_2}$ is set as explained in Section 5.3.2 for a buffer zone of five meters separating the stations from the storage area. The number of stations $|\mathcal{S}|$ is between two to four. The number of pods is equal to 85% of the number of storage locations, i.e., $|\mathcal{P}| = 0.85|\mathcal{L}|$. The number of distinct products available in the storage area $|\mathcal{I}|$ is between 10 to 500. Being $\mathcal{I}_p \subseteq \mathcal{I}$ the set of products contained in a pod p, we set $|\mathcal{I}_p|$ to be between 3 to 25 products. These combinations allow us to analyze different levels of products scatteredness in the storage area. All instances have a planning horizon of two waves. Limiting $|\mathcal{W}| = 2$ has the advantage of requiring less computational effort to solve instances and allowing a fair comparison between our solution approaches with a smaller set of instances. New waves are triggered when there are enough products in the backlog to use most of the capacity available to reduce pickers' idle time, without overloading the system and leaving some flexibility to move orders between stations when solving the OAP. The orders in each wave are generated using the procedure described in Section 5.6.2 for the scenarios generation such that each order has one to four products, the average order comprises 1.6 items, and the majority of them has a single item, as seen in e-commerce distribution centers. The initial location of pods L_p is always determined randomly. Stations capacities C range between 3 to 15 orders. The maximum difference of order lines picked by stations δ is arbitrarily fixed at four to allow some flexibility when assigning orders to stations. Finally, three demand skewness are considered to generate orders in all sizes, ranging from 20% of orders accounting for 80% (low), 50% (medium), and 33% (high) of all demands. The given parameters result in three settings for the tiny layout and 12 settings for the remaining layouts each. Random seeds were used to generate 20 instances for each setting, which result in 780 instances in total, all of them carefully generated to contain feasible solutions.

5.6.2 Scenarios generation

A scenario contains a certain number of orders, each with a certain number of products, which can be a combination of any subset of products among all products available $\mathcal{I}$.

The first decision when sampling a scenario to estimate the stochastic costs is to determine the number of orders to be sampled. Since the starting point of a wave is a decision controlled by the warehouse manager, we simply consider that the number of orders to be generated is fixed to the number of orders arrived in the current wave.

The next decision is about the number of order lines contained in each order. Orders in e-

commerce are typically composed of few products. The number of order lines in each order is commonly generated from a truncated geometric distribution (Lamballais et al., 2017; Valle and Beasley, 2021; Xie et al., 2021). In this distribution, given a parameter μ the probability that an order contains m distinct products is represented by $\mu(1-\mu)^{(m-1)}/\sum_{n=1}^{M}\mu(1-\mu)^{(n-1)}$ for $m = \{1,\ldots,M\}$, where M is the maximum number of items in an order. Following Valle and Beasley (2021), we generated orders using $M = 4$ and $\mu = 1/1.73$, chosen such that orders have an average of 1.6 order lines, which is known to be the average order demand at German Amazon warehouses (Boysen et al., 2019b). The chosen distribution also implies that 85% of orders contain only one item.

There is only left to determine the lines of each order, which can be generated using an ABC curve (Caron et al., 1998). The ABC curve is used to represent demand skewness by a continuous analytical function. The more skewed demands are, the more weight few products have in the total demand. The ABC curve is given as

$$F(x) = \frac{(1+s)x}{s+x}, \quad 0 \leq x \leq 1, s \geq 0, s+x \neq 0, \tag{5.39}$$

where x indicates the relative position of a product whose order frequency represent a fraction $F(x)$ of total warehouse activity. The parameter s indicates the skewness of the demand. In the instances generated, we used $s = 0.067$ to represent the low skewness case, holding that that 20% of the products ($x = 0.2$) account for 80% of the picks ($F(x) = 0.8$), reducing to 50% when $s = 0.333$ in the medium skewness case, and to 33% when $s = 1$ in the high skewness case. Given that the available products $\mathcal{I}$ are sorted by their demands, the ABC curve is used to generate the order lines for the scenarios from random x values. Finally, equal probabilities of occurrence are assumed for each scenario among the S scenarios sampled, i.e., $P(\xi) = 1/S$.

5.6.3 Comparisons for smaller instance sets

We start our analysis by using all methods presented, i.e., all exact approaches and our matheuristic, to solve the smaller instance sets. Thus, we can observe what instance sizes each method is capable of solving within a reasonable time, defined here to be a maximum of one hour for each run. Four scenarios are sampled for the stochastic approach and the matheuristic to estimate the stochastic costs, which is the number of cores available in our computational environment, to achieve high efficiency. Table 5.5 compares the solutions obtained when solving the sets *Tiny* to *Medium*. Column *Opt* shows the percentage of instances solved to optimality within the time limit established. This indicator is not shown for the matheuristic since optimality is not proven when using it. Column *Time (s)* is the average time in seconds to prove optimality using the exact models or to stop the matheuristic when it is the case. Columns *Bring pods*, 1^{st} *wave*, and *All waves* show the average energy cost to bring pods to stations in the first wave, the average total cost for the first wave, and the average total cost for both waves, respectively, for the instances solved within the time limit. We highlight in

the table the best costs for each one to stress that the sequential approach optimizes the cost to bring pods in the 1^{st}*wave*, the myopic approach optimizes all the costs for the 1^{st} wave, and the optimal approach optimizes all the costs for the whole planning horizon.

Table 5.5 – Methods comparison for the smaller instance sets

Approach	*Layout*	*Opt*	*Time (s)*	*Energy cost*		
				Bring pods	1^{st} *wave*	*All waves*
Optimal	*Tiny*	100%	85.8	9.99	20.04	**39.38**
	Small	0%	>3600	–	–	–
	Medium	0%	>3600	–	–	–
Stochastic	*Tiny*	97%	155.0	9.89*	19.09*	39.09*
	Small	0%	>3600	–	–	–
	Medium	0%	>3600	–	–	–
Myopic	*Tiny*	100%	0.3	9.78	**19.45**	39.71
	Small	100%	34.0	14.99	**28.17**	61.73
	Medium	0%	>3600	–	–	–
Sequential	*Tiny*	100%	0.1	**9.68**	19.68	39.87
	Small	100%	0.2	**14.43**	29.22	65.06
	Medium	100%	4.8	**30.47**	61.27	132.60
Matheuristic	*Tiny*	–	0.3	9.76	19.62	39.62
	Small	–	1.5	14.80	28.53	62.19
	Medium	–	26.1	30.72	60.46	129.28

*: calculated only for the instances with optimality proven

Some observations are made from Table 5.5. First, the optimal and stochastic approaches cannot solve any instance larger than those in the *Tiny* set, while the myopic approach cannot solve any instance larger than those in the *Small* set, which significantly limits the applicability of these methods in practice. From the instances with optimality proven, we note that the myopic approach is leading to solutions on average 1.1% (*Tiny*) and 3.6% (*Small*) better than the sequential approach for the current wave being planned, which attests the effectiveness of integrating PRP decisions with the OAP and the PSP. However, solutions found by the myopic approach are still 0.8% (*Tiny*) further than the best possible solutions as given by the optimal approach. This means that there is room for improvement that could be reached using the stochastic approach. Unfortunately, the stochastic approach could not solve all instances from any sets, so a direct comparison between the costs shown in the table against any other approach is not possible. Removing the instances that the stochastic approach could not solve from the other approaches, we still observe a slight improvement of 0.04% compared to the myopic approach and a distance of 0.69% to the optimal approach.

In summary, our matheuristic is significantly improving the initial solutions generated by solving the OAP–PSP model from the sequential approach. For the *tiny* instances, we see an improvement of the average solution compared to the myopic approach, which indicates that the SAA technique used is being somewhat effective. For the *small* set, although solutions are

not better in the matheuristic than in the myopic approach, we can see that it is approximating them well in considerably less time. This time advantage becomes clear in the *medium* set when the myopic approach is not able to prove the optimality of any instances while the matheuristic is quickly finding improvements for the initial solutions provided by the sequential approach.

5.6.4 Comparisons for the largest instance set

Now, we conduct a deeper analysis for the *large* instance set using the two methods – the sequential approach and the matheuristic – capable of solving instances at this size within the same time limit of one hour. The results are presented in Table 5.6, detailed for the number of products ($|\mathcal{I}|$), the number of products per pod ($|\mathcal{I}_p|$), and the demand skewness (*skew*). Column *#Inst* shows the number of instances – out of a total of 20 instances – solved to optimality for the sequential approach and that the matheuristic finished its run within the time limit. *Time (s)* reports the average run time, and *Cost* is the equivalent of the column *All waves* from Table 5.5. We also display a column *Diff.* showing how much our matheuristic is improving the initial solution generated by the sequential approach. The improvements are calculated only for the instances finished within the time limit using both methods.

Table 5.6 – Methods comparison for the *large* instance set

$\|\mathcal{I}\|$	$\|\mathcal{I}_p\|$	*skew*	*Sequential approach*			*Matheuristic*			
			#Inst	*Time (s)*	*Cost*	*#Inst*	*Time (s)*	*Cost*	*Diff.*
200	*10*	*33*	20	220.6	319.96	20	965.2	316.15	-1.19%
		50	20	166.7	309.52	20	649.5	308.85	-0.22%
		80	20	76.7	279.09	20	744.0	276.11	-1.07%
	25	*33*	19	907.2	178.65	2	2938.4	180.26	1.11%
		50	19	636.2	177.15	9	2593.1	168.86	-1.17%
		80	20	367.5	165.70	14	1984.8	164.24	0.54%
500	*10*	*33*	20	6.5	599.18	20	52.4	595.03	-0.69%
		50	20	8.0	573.99	20	57.4	572.71	-0.22%
		80	20	11.6	537.62	20	56.1	532.04	-1.04%
	25	*33*	20	302.1	334.07	20	1273.7	331.17	-0.87%
		50	20	269.2	323.70	19	1208.3	320.18	-0.83%
		80	20	175.8	302.97	20	778.1	298.19	-1.58%

From Table 5.6, we see that in most cases our local search matheuristic improves the solutions for the sequential approach for the real costs observed after all picking waves. This is a clear sign that solving the stochastic model – approximated here by our matheuristic – using the SAA scheme presented would lead to better solutions for the real demands than using the sequential approach. The average improvement observed is by 0.76% for those real-size instances. Instances with a lower $|\mathcal{I}|$ and a higher $|\mathcal{I}_p|$ have products more scattered within the storage area and, consequently, are harder to solve due to the larger number of pod options to choose to carry to stations. Given 428 pods in this layout size, when $|\mathcal{I}| = 200$ and $|\mathcal{I}_p| = 25$

each product can be found on average in 53.5 pods. Meanwhile, when $|\mathcal{I}| = 500$ and $|\mathcal{I}_p| = 10$ each product is stocked only in 8.6 pods on average. Despite the increased difficulty to solve, more scattered storage leads to much lower energy consumption, reducing from nearly 600 kJ per wave to nearly 180 kJ per wave comparing the two most extreme situations investigated and a high demand skewness. Another conclusion drawn here is that energy consumption is reduced when the demands are more skewed. Reduction is around 10.6% to 12.7% when the 20% most demanded products account for 80% of the total demand instead of only 33%.

We highlight that not all runs of the sequential approach are being finished within the time limit, indicating that this is approximately the largest instance size this method can be used in practice. Since the matheuristic starts from this solution, we cannot use it to solve larger instances either, unless a low scattered storage level is used, which is not common in an RMFS.

5.6.5 Further analysis

In this section, we extend our analysis to two new cases that are worth investigating since they can significantly impact energy consumption in an RMFS. In the first case, we show how our models can be used to minimize the number of pod visits, which is a common performance measure optimized in the RMFS literature, as mentioned before, and compare solutions for this metric and the energy consumption when either of the two are minimized. The second case presents a situation where picks can be delayed so that waves are planned after the backlog have more orders than the scenarios considered so far. We compare the results for a backlog with twice more orders to show that, whenever possible, delaying picks can significantly save energy due to a more efficient order assignment solution.

Minimize the number of pod visits versus energy consumption

Our exact models can be easily modified to optimize the number of pod visits instead of the energy consumed by robots. For that, it is enough to change the objective function (5.20) for the integrated OAP–PSP to

$$\min \sum_{p \in \mathcal{P}} \sum_{s \in \mathcal{S}} x_{ps}. \tag{5.40}$$

We run the modified model to analyze the trade-off between energy consumption and the number of pod visits when solving the same model for each objective, solving the PRP using the nearest rule for both cases. Table 5.7 summarizes the results obtained. We removed from this analysis all instances that took longer than one hour in either of the models to allow a direct comparison of the results. The number of instances compared is shown by *#Inst.* Columns *Energy* and *#Pods* show, respectively, the average energy consumption and the number of pod visits for the solutions found. Columns *Diff.* shows the difference in each indicator when solving the problem for minimizing the number of pod visits compared to when minimizing energy consumption. Overall, the results show that minimizing the number

of pod visits leads to solutions with a 9.1% to 18.6% higher energy consumption, even though between 11.7% and 25.2% fewer pods are carried to stations. Despite the savings in energy consumption, it is possible that blocking becomes more frequent, and long queues are formed by the stations when more pods are carried around, which may affect energy consumption in practice. Also, larger costs can occur from the larger number of robots required to pick the extra pods. These drawbacks should be weighted by the warehouse manager when deciding which objective function to be minimized.

Table 5.7 – Comparison between minimizing energy consumption and minimizing the number of pod visits

$\|\mathcal{I}\|$	$\|\mathcal{I}_p\|$	*skew*	*#Inst*	*Min energy*		*Min #Pods*			
				Energy	*#Pods*	*Energy*	*Diff.*	*#Pods*	*Diff.*
200	*10*	*33*	20	319.96	33.8	367.33	12.9%	27.9	-21.0%
		50	20	309.52	32.8	359.88	14.0%	27.2	-20.6%
		80	20	279.09	29.5	316.23	11.7%	24.2	-21.9%
	25	*33*	16	176.71	20.6	217.04	18.6%	16.9	-22.2%
		50	18	176.48	20.5	211.04	16.4%	16.3	-25.9%
		80	20	165.70	19.1	190.90	13.2%	15.3	-25.2%
500	*10*	*33*	20	599.18	54.1	659.75	9.2%	47.4	-14.1%
		50	20	573.99	51.2	631.50	9.1%	45.9	-11.7%
		80	20	537.62	48.9	596.46	9.9%	42.9	-13.9%
	25	*33*	20	334.07	33.8	379.45	12.0%	28.2	-19.7%
		50	20	323.70	32.7	371.81	12.9%	27.7	-17.9%
		80	20	302.97	31.0	341.45	11.3%	26.1	-18.8%

Picking orders immediately versus waiting until all orders arrive

Normally, the longer we wait to make decisions, the more information becomes available and the more efficient order picking can be. We present an alternative approach to solve the integrated OAP–PSP–PRP for when the time available for the picks is not tight, such that we can delay the picks to be done after more orders arrive than the available capacity. We call this a *wait-and-see approach*. This approach can be seen as a myopic approach with a larger number of orders in the backlog. In practice, it is preferred to wait for orders to be picked in a later period with no significant penalty in the demand satisfaction, such as when dealing with low-priority orders. In this case, more orders in the backlog allow the picks to be planned more efficiently.

We solved all instances for the large layout set, either picking orders when arrive or using the wait-and-see approach. Despite the possibility of using the INLP model for the myopic approach to solve the wait-and-see approach, we opted to use our matheuristic in this analysis so that we could compare solutions for the largest instance set, which is closer to found in a real situation. We solve it considering that the orders for both waves are ready for picking at the beginning of the first picking wave. Then, we run our matheuristic limiting the number of

picks to be half the number of orders in the backlog for a fair comparison against solving the original problem. After solving the problem for the first wave, we remove the orders picked and run again the matheuristic for the remaining orders. Since the wait-and-see approach generalizes the myopic approach, an optimal solution for it is a lower bound for the sum of the optimal solutions of the myopic approach for each wave. In practice, the wait-and-see approach may allow new orders to be added to the backlog as they arrive. The results found show that the wait-and-see approach leads to solutions between 16% to 17.8% cheaper than picking as orders arrive for the instances tested. The major drawback of this approach is the increase in order cycle times. Again, these have to be weighted when deciding which approach to use in practice.

5.7 Conclusions

In this paper, we investigated how the repositioning of inventory pods in the robotic mobile fulfillment system can lead to a more efficient order picking process. This is analyzed by observing the energy consumption reduction of robots carrying pods between the storage area and the picking stations. We integrate pod repositioning decisions with those for two other operational problems commonly found in this system, namely the order assignment and the pod selection.

We proposed several approaches to solve the integrated problem using a wave picking strategy for when future demands are uncertain. We showed that when pod repositioning decisions are integrated with other decisions, waves can be performed consuming up to 3.6% less energy than when decisions are made sequentially. When we add stochastic information about future demands by sampling a few scenarios for them to the decision process, solving a two-stage stochastic programming model, solutions can be improved even further. We still show that these solutions are only 0.69% on average below the best possible case when future demands are known.

We presented a local search matheuristic that starts from a solution generated by the sequential approach and improves it by searching a neighborhood with solutions where pods are returned to different locations after the pickings are done. Our matheuristic also uses information about future demands to provide robust solutions for the integrated problem. Our experiments showed that this matheuristic can find solutions up to 1.58% better than the sequential approach when instances are considerably larger than those solved by the exact methods.

Finally, further analysis showed how to adapt our methods to minimize the number of pod visits instead of energy consumption. Our experiments showed that minimizing pod visits can lead to solutions with up to 18.6% higher energy consumption compared to when energy consumption is explicitly minimized. A second case analyzed is when orders can wait to be

picked in a later wave. We showed that a backlog with twice more orders can reduce energy consumption by up to 17.8%.

Conclusion

Designing storage and picking systems in warehouses, from the traditional manual system to the innovative robotic mobile fulfillment system, requires a series of important decisions that highly impact their performance. The relationship between storage and picking processes is a growing topic of research due to the current economic context, mainly with the quick rise of e-commerce. Also, the practical implications that designing these processes have when using such warehousing systems is a topic of interest. Tactical and operational level decisions regarding the storage, batching, and routing are among those with higher interest since they are responsible for the majority of operating costs in a warehouse. Traditionally, these problems are solved using simple policies that present guidelines that can be easily followed by pickers and managed by warehouse managers since they circumvent dealing with the stochastic nature of processes. Recent advances in solution techniques and advances in computational power allow that traditional problems found when designing storage and picking systems are reanalyzed and solved in a more integrated manner to consider the characteristics of the warehouse in question. In this thesis, we have introduced, modeled, and solved storage, batching, and routing problems found in the manual system and the RMFS, providing new tools to those traditionally found in the warehousing literature and used in practice.

In Chapter 1, we introduced the basic concepts of the storage and picking systems in warehouses, showing that they can be viewed as an integrated system in practice due to their relationship. Two S/P systems are described: the picker-to-product manual system and the product-to-picker RMFS. The most common storage, batching, and routing decision problems in these systems were introduced, and the solution techniques found in the literature were described. These problems are usually solved sequentially using simple policies developed for each individual problem. Our review showed that most of the studies found that attempt to evaluate the interactions between these problems focus on answering which set of policies result in the best performance when combined for different warehouse settings. Decisions are more commonly integrated between batching and routing. However, integrating storage decisions with them can lead to significant improvements in many performance measures as shown by the studies that already investigated their interactions. To fill this gap in the literature, all studies presented in this thesis either suggest new models or improve the known ones to solve storage problems found in the manual and RMFS. This is done considering different

warehouse settings and the batching and routing decisions made.

In Chapter 2, we integrated the storage and routing decisions in a manual warehouse. We considered that a set of future orders is known, each representing a pick list to be followed by pickers. The set of products to be picked should be arranged in the storage area such that the total distance traveled by pickers to perform all routes is minimized. A general model for the problem was introduced that does not consider any specific warehouse layout. Four other models were presented for the cases that the routing problem must be solved using popular policies for the single-block layout. From computational experiments, we showed that these models can only be solved for small instances within a reasonable time. Therefore, we proposed a general variable neighborhood search such that larger size instances, such as those found in practice, can be solved faster. Our results revealed that the commonly used routing policies can effectively speed up the search for an optimal assignment of products to storage locations when using the proposed metaheuristic. Also, we showed that these near-optimal solutions are a considerable improvement compared to those generated from the common storage policies used in practice.

In Chapter 3, we relaxed the assumption that all future orders are fully known to solve the storage problem in a manual warehouse. In this case, this problem is commonly solved using the ABC storage policy, which divides the storage area into three zones and assigns the products with the highest expected demands to the best zones. The optimal size of each zone is known to be influenced by several factors, such as the warehouse layout, the demand characteristics, and the policies used to define the zones shapes and picking routes. We implemented a simulator to estimate the average route length given a huge number of combinations of these factors and zone sizes. A grid search was performed to find the zone sizes that resulted in the lowest expected route length for each combination of factors. From these findings, we evaluated which factors impact the most the choice of good zone sizes using regression analysis. In practice, zone sizes are commonly chosen using fixed values as a rule-of-thumb. We trained four machine learning tools – ordinary least squares, regression tree, random forest, and multilayer perceptron – using the data generated from the simulations to suggest well-performing zone sizes from the most important factors observed. These tools were evaluated and compared against the arbitrarily fixed sizes and the simulator, both quantitative and qualitatively. The results showed that even the simpler tools could suggest zone sizes that perform better than the common arbitrary sizes, and they can be significantly easier to be implemented in practice than the simulator.

If enough data is available to identify correlations between products, a different strategy to solve the storage problem is by locating correlated products close to each other, in a manual system, or together in the same pod, in an RMFS. This may lead to significant reductions in the travel time when these products are ordered together. A common way to model the storage problem using a correlated storage policy is as a quadratic assignment problem. In Chapter

4, we investigated the QAP and four of its most common variants found in the literature. These problems were modeled as non-linear integer programming models and linearized using common linearization techniques. Computational experiments showed that these models can only solve instances of a limited size. Thus, we proposed a parallel memetic iterated tabu search as an alternative to solve larger instances faster. The results attested to the effectiveness of the parallelism since the metaheuristic was able to find or improve the majority of best known solutions of the benchmark instances tested. Our parallel metaheuristic significantly updated the literature of the QAP variants and can be easily adapted to solve other variants, including those used to solve the storage problem with a correlated storage policy.

In Chapter 5, we integrate storage, batching, and routing decisions for an RMFS. Most of the RMFS literature focuses on solving these problems sequentially and using an online picking strategy, such that the throughput rate of the warehouse is maximized disregarding the operational costs to achieve this rate. The use of a wave picking strategy can lead to a more efficient operation since multiple orders can be batched to be picked together, reducing the number of robots traveling with pods within the warehouse. We analyzed the impact of integrating pod repositioning decisions with order assignment and pod selection for the energy consumption in a typical RMFS. Due to the dynamic nature of wave picking, we suggested using a two-stage stochastic programming model to account for the uncertainty of future demands when planning a picking wave. A local search matheuristic derived from the mathematical models presented was used to solve real size instances. Our experiments showed that the sampling scheme used to simulate future demands is effective to reposition pods after they are requested within the storage area accounting for the expected demands in future waves. We still showed that explicitly minimizing energy consumption can lead to a significant improvement in the picking efficiency than minimizing the number of pods visits, as commonly done in the literature.

This thesis contributes to the literature by presenting many modern optimization techniques to support the decision process when designing warehousing systems, specifically for the storage and picking processes. Overall, our research shows that there is still a large gap between the methods commonly used in practice to solve the many tactical and operations problems studied and those found in the warehousing literature. Despite warehouses being a well-established component of supply chains, the many different characteristics found in these facilities in practice is still an obstacle to generalizing the analyzes done in this and other studies. Therefore, a limitation of our studies is that practical confirmation of our conclusions is still lacking. We understand that using the suggested techniques has practical limitations due to the inherent complexity to solve most of the problems found. As our experiments show, despite the clear contributions to the existing literature, many of our models could barely handle real size instances even by powerful computational machines, which in most cases are too expensive to be used in warehouses. The stochastic nature of operations further

limits their application in practice. Therefore, it is more practical to implement simple policies that are proven to perform slightly well for the average case than tailoring solutions for each specific case in most situations. As a future research direction, we observed that there is still much room to investigate different warehousing systems and their implications when adopted by warehouses used in different markets. A major challenge is to consider the limitations found in practice to make the developed methods easier to be adopted.

Appendix A

Appendices of chapter 2

A.1 Neighborhood exploration order

Table A.1 presents the results for all rounds of experiments with different neighborhood exploration orders. Column *LS order* shows the exploration order of the neighborhoods. The subsequent columns show the average gap of the VND solution to the optimum found by the MIP for each of the five problems for the instances solved up to optimality with the best gaps in bold, and column *Avg.* is the weighted mean of the gaps with the number of instances in each problem as weights. To compare, we show the average gap of the three initial solutions generated by each of the storage policies to the optimal solution found by the MIP in row *Initial solution avg. gap (%)*. Computation times are not reported since all VND calls took less than a second to finish.

Table A.1 – Results for the exploration of different combinations of neighborhoods in a VND

LS order	*SLOPP* Gap (%)	*SL+Re* Gap (%)	*SL+Ss* Gap (%)	*SL+Mp* Gap (%)	*SL+Lg* Gap (%)	*Avg.*
Slot	**0.00**	5.84	8.50	3.53	6.67	6.02
Row	9.06	19.44	21.67	16.22	19.55	18.95
Aisle	21.97	33.38	36.01	27.70	33.31	32.40
Slot → *Row*	**0.00**	4.16	7.50	2.41	5.81	4.89
Slot → *Aisle*	**0.00**	5.61	6.47	3.39	4.68	4.93
Row → *Slot*	**0.00**	4.51	7.45	2.91	5.76	5.06
Row → *Aisle*	8.51	18.71	20.57	15.49	18.44	18.04
Aisle → *Slot*	**0.00**	5.89	6.91	3.47	4.49	5.10
Aisle → *Row*	8.51	18.54	20.67	15.15	18.04	17.86
Slot → *Row* → *Aisle*	**0.00**	3.91	5.49	2.34	3.80	3.82
Slot → *Aisle* → *Row*	**0.00**	**3.90**	5.51	2.34	3.84	3.83
Row → *Slot* → *Aisle*	**0.00**	4.28	**5.37**	2.80	3.91	4.00
Row → *Aisle* → *Slot*	**0.00**	4.26	5.53	2.78	3.95	4.04
Aisle → *Slot* → *Row*	**0.00**	3.91	5.50	**2.17**	**3.66**	3.76
Aisle → *Row* → *Slot*	**0.00**	4.09	5.46	2.82	3.76	3.95
Initial solution avg. gap (%)	22.33	33.61	36.45	28.19	33.84	32.81

A.2 Setting the maximum number of shakes

We test S for different values to find if there is an appropriate one that leads to better solutions. The GVNS was tested with $S = \{1, 2, 4, 5, 10, 20, 40\}$. Since VND is the most costly step in our algorithm, we decided to use $K = 400/S$ in order to keep the number of VND calls approximately constant in each run. Each instance of the small set is solved 10 times, starting from each of the three storage policies solution for each one of the five problems considered. Table A.2 reports the results found. They are divided into two sets. *Optimal* is for the instances solved up to optimality by CPLEX, while *Feasible* is for those with an upper bound solution known but with optimality unproven within the time limit of two hours. Results for the integrated SLOPP solved using the GVNS scheme are not reported since the optimal was already found by only using a single VND, as shown previously in Table A.1, and there is not a significant number of instances in the *Feasible* category for this problem. The results show that the GVNS performance for $S = 5$ is, on average, better than for other values.

Table A.2 – Results for the parameter tuning of the maximum number of shakes S

	S	K	*SL+Re*		*SL+Ss*		*SL+Mp*		*SL+Lg*	
			Gap (%)	*Time (s)*	*Gap (%)*	*Time (s)*	*Gap (%)*	*Time (s)*	*Gap (%)*	*Time (s)*
Optimal	1	400	0.03	0.4	0.02	0.4	0.00	0.3	0.01	0.5
	2	200	0.02	0.4	0.00	0.5	0.00	0.3	0.00	0.5
	4	100	0.01	0.4	0.00	0.5	0.00	0.4	0.00	0.6
	5	80	0.00	0.5	0.00	0.5	0.00	0.4	0.00	0.6
	10	40	0.00	0.5	0.00	0.6	0.00	0.4	0.00	0.7
	20	20	0.00	0.5	0.00	0.6	0.00	0.4	0.00	0.7
	40	10	0.00	0.6	0.00	0.7	0.00	0.5	0.00	0.8
	#Inst		92		99		75		80	
Feasible	1	400	-0.28	1.5	0.99	1.9	-3.34	1.6	-5.70	2.5
	2	200	-0.40	1.7	0.58	2.1	-3.50	1.8	-5.97	2.7
	4	100	-0.52	1.8	0.43	2.4	**-3.59**	1.9	**-6.12**	3.1
	5	80	**-0.58**	1.9	**0.35**	2.5	**-3.59**	2.0	-6.11	3.2
	10	40	-0.42	2.2	0.49	2.9	-3.58	2.3	-6.09	3.8
	20	20	-0.36	2.5	0.63	3.5	-3.53	2.6	-5.95	4.5
	40	10	-0.32	2.7	0.95	4.1	-3.49	2.9	-5.77	5.2
	#Inst		16		9		33		28	

A.3 Average time and number of cycles when solving the regular set

Table A.3 complements Table 2.5 by presenting the average running time of GVNS when stopped by one of the stopping criteria (7200 seconds or $K = 10,000$ cycles) and the last cycle k performed during the search. Time may be above 7,200 seconds due to the placement of the verification of the time stopping condition in the algorithm. Rows that neither show a time of 7200 seconds nor $k = 10,000$ indicate that instances within the group were stopped by different criteria between both.

Table A.3 – Average running time and last cycle performed by GVNS for the SLOPP with different routing policies

A	B	$\mathcal{O}$	$\mathcal{Q}_o$	*Re*		*Ss*		*Mp*		*Lg*		*LKH*	
				Time (s)	k	*Time (s)*	k	*Time (s)*	k	*Time (s)*	k	*Time (s)*	k
5	10	10	5	374	10000	504	10000	374	10000	406	10000	7201	736
			20	1379	7858	2251	10000	1379	10000	1547	10000	7226	24
			50	3411	2534	5546	10000	3411	10000	3059	10000	7335	3
		30	5	1110	10000	1598	10000	1110	10000	1358	10000	7203	215
			20	4626	2337	7074	9894	4626	10000	5121	10000	7278	5
			50	7200	750	7200	3361	7200	6133	7200	7367	7705	1
		50	5	1868	9976	2712	10000	1868	10000	2236	10000	7204	119
			20	7200	1314	7200	5521	7200	8790	7200	8353	7353	2
			50	7200	413	7200	1682	7200	3450	7200	4181	8239	1
	50	10	5	5220	10000	5858	10000	5220	10000	4537	10000	7204	135
			20	7200	1699	7200	4743	7200	6086	7200	6474	7899	1
			50	7200	404	7201	1549	7200	1891	7200	2436	7577	1
		30	5	7200	3766	7200	4566	7200	5432	7200	6030	7220	31
			20	7201	438	7201	1278	7201	1626	7200	1865	7692	1
			50	7201	78	7201	367	7201	457	7202	613	7200	1
		50	5	7200	1861	7200	2616	7200	3143	7200	3151	7227	15
			20	7201	196	7201	654	7201	885	7201	949	7262	1
			50	7203	32	7204	160	7203	252	7202	312	7200	1
10	10	10	5	1095	10000	1401	10000	1095	10000	1106	10000	7202	324
			20	3316	2533	5829	10000	3316	10000	4157	10000	7267	7
			50	7200	595	7200	4967	7200	6754	7200	6505	7750	1
		30	5	2883	8218	3964	10000	2883	10000	3440	10000	7206	95
			20	7200	674	7200	3808	7200	6508	7200	5403	7431	1
			50	7200	162	7201	1206	7200	2040	7200	1910	7335	1
		50	5	4834	4182	6905	9899	4834	10000	5913	10000	7210	52
			20	7200	390	7200	2152	7200	3670	7200	3118	7618	1
			50	7201	88	7201	588	7201	1103	7201	1131	7201	1
	50	10	5	7200	3692	7200	3517	7200	3767	7200	4529	7213	62
			20	7200	681	7200	1909	7200	2333	7200	2220	8106	1
			50	7201	154	7201	699	7201	931	7201	875	7683	1
		30	5	7200	1504	7200	1534	7200	1717	7200	2049	7235	17
			20	7201	196	7201	562	7201	741	7201	684	7430	1
			50	7203	14	7204	158	7203	219	7203	243	7202	1
		50	5	7201	805	7201	896	7201	1017	7201	1155	7259	8
			20	7201	82	7203	301	7201	385	7202	382	7200	1
			50	7205	2	7208	48	7205	95	7204	127	7201	1
20	10	10	5	3692	10000	4442	10000	3692	10000	3376	10000	7205	156
			20	7200	1184	7200	4972	7200	7989	7200	6906	7435	2
			50	7200	182	7200	1742	7200	2611	7200	2335	7222	1
		30	5	7200	3791	7200	6511	7200	8381	7200	7662	7212	45
			20	7200	238	7200	1468	7200	2746	7200	2210	7966	1
			50	7201	38	7201	413	7201	828	7201	669	7209	1
		50	5	7200	1882	7200	3704	7200	5026	7200	4318	7232	20
			20	7200	117	7201	803	7200	1550	7201	1220	8117	1
			50	7201	17	7203	202	7201	432	7202	385	7206	1
	50	10	5	7201	1074	7201	925	7201	962	7200	1211	7225	30
			20	7201	273	7201	540	7201	635	7201	675	8553	1
			50	7202	74	7203	247	7202	324	7202	312	7201	1
		30	5	7201	497	7201	438	7201	474	7201	599	7272	8
			20	7202	88	7203	222	7202	282	7202	218	7208	1
			50	7206	5	7211	67	7206	94	7207	80	7205	1
		50	5	7202	276	7203	267	7202	289	7202	342	7347	4
			20	7204	33	7205	112	7204	149	7205	115	7211	1
			50	7212	1	7217	18	7212	30	7210	36	7214	1
Average				6227	4115	6272	4099	6492	3543	6873	2174	7393	40

A.4 Results for the SLOPP using GVNS with the Mp/LKH search strategy

Table A.4 presents the results for the experiments with GVNS using the combined Mp/LKH routing creation strategy as described in Section 2.5.3 for the instances of the regular set.

Table A.4 – Results for the SLOPP using GVNS with the Mp/LKH search strategy

B	$\mathcal{O}$	$\mathcal{Q}_o$	*A = 5*			*A = 10*			*A = 20*		
			Avg. sol.	*Time (s)*	*k*	*Avg. sol.*	*Time (s)*	*k*	*Avg. sol.*	*Time (s)*	*k*
		5	108.9	539	10000	110	1485	10000	112.7	4637	10000
	10	20	292.3	2350	10000	319.6	5971	10000	336	7200	4921
		50	521.3	5585	10000	662.5	7200	5073	724.6	7201	1816
		5	496.8	1719	10000	512	4234	10000	521.6	7200	6090
10	30	20	1136.7	7146	9888	1437.3	7200	3878	1624.4	7200	1507
		50	1741.5	7200	4537	2512.7	7200	1662	3277.6	7201	531
		5	966	2893	10000	1049.3	7090	9589	1085.2	7200	3528
	50	20	2050.1	7200	5800	2707.4	7200	2268	3225.7	7201	842
		50	2997.4	7200	2578	4523.8	7201	865	6207.6	7203	279
		5	113.2	6164	10000	114.9	7200	3347	116.8	7201	880
	10	20	374.3	7200	4314	324.8	7200	1800	321.7	7201	524
		50	951.7	7201	1488	809.5	7201	649	790.4	7202	232
		5	582.7	7200	4286	543.2	7201	1400	557.6	7202	398
50	30	20	2383.6	7201	1227	1877	7201	523	1638.6	7203	208
		50	4039.2	7202	406	4557.1	7204	164	4536.8	7209	69
		5	1297.8	7200	2417	1152.7	7201	825	1161.8	7202	246
	50	20	4902	7201	635	4897.9	7203	291	3698.4	7205	104
		50	7795.5	7204	194	8931.1	7208	57	9538.6	7222	21

*Avg. sol. = 2023.5; Avg. time = 6524s; Avg. k = 3562

Appendix B

Appendices of chapter 3

B.1 Detailed results by S/R policy

Table B.1 – Average and maximum percentage deviations to the ARL of the BKS found in the simulations for each S/R policy combination (best values in boldface)

S/R Policy	*18/35/47*		*Avg per S/R*		*OLS*		*RT*		*RF*		*MLP*	
	Avg (%)	Max (%)	Avg (%)	Max (%)	Avg (%)	Max (%)	Avg (%)	Max (%)	Avg (%)	Max (%)	Avg (%)	Max (%)
Aa/Aba	1.55	4.30	1.44	8.58	0.63	6.75	0.34	3.41	**0.27**	**1.42**	0.28	2.02
Aa/Sh	0.99	4.35	0.85	6.45	0.75	5.52	**0.23**	**2.59**	0.29	5.44	0.37	6.21
Aa/Lg	2.08	5.47	1.81	6.08	1.60	5.46	**0.53**	5.33	0.66	**4.74**	0.61	5.17
Aa/Co	1.16	3.77	1.22	4.29	0.94	3.64	0.42	5.50	**0.30**	**2.57**	0.36	2.74
Nl/Aba	0.76	5.22	0.81	6.88	0.49	3.21	0.44	2.28	**0.35**	**2.28**	0.38	2.93
Nl/Sh	0.96	5.24	0.91	4.90	0.69	4.01	0.61	2.83	0.46	2.74	**0.45**	**2.73**
Nl/Lg	0.98	5.97	0.87	4.22	0.62	3.45	0.58	**3.44**	**0.47**	3.63	0.49	3.52
Nl/Co	0.93	5.79	0.87	4.94	0.63	3.70	0.48	2.43	**0.38**	**1.90**	0.39	2.58
Ns/Aba	1.31	7.64	1.45	9.55	1.03	6.15	1.04	**4.65**	**0.85**	5.62	0.91	5.64
Ns/Sh	1.56	8.75	1.47	8.13	1.19	5.42	1.22	5.23	1.01	**4.18**	**0.96**	4.58
Ns/Lg	1.57	7.26	1.41	6.71	1.32	6.64	1.20	4.73	**1.06**	5.25	1.08	**4.29**
Ns/Co	1.47	6.53	1.40	5.96	1.23	5.68	1.15	5.64	**0.91**	**4.77**	1.02	6.12
Wa/Aba	2.24	14.01	2.14	15.43	1.64	10.17	1.15	**5.98**	**1.01**	6.36	1.10	7.52
Wa/Sh	1.88	10.00	1.73	7.92	1.20	7.39	0.96	5.21	0.85	3.85	**0.66**	**3.48**
Wa/Lg	2.51	10.11	1.87	11.22	1.60	18.60	0.89	7.07	0.69	6.30	**0.60**	**3.13**
Wa/Co	1.97	10.50	1.71	6.84	1.30	17.94	0.91	4.83	0.80	4.98	**0.70**	**4.68**
Average	1.50	7.18	1.37	7.38	1.05	7.11	0.76	4.45	**0.65**	**4.13**	**0.65**	4.21

Appendix C

Appendices of chapter 4

C.1 Comparison of different models for the QAP and its variants

We performed a set of experiments using the quadratic integer programming (QIP) model and two linearization techniques – the standard linearization technique (Standard) and the level-1 reformulation linearization technique (RLT-1) – for all benchmark instance sets for each problem to verify the best performing model among the three to be solved. We note that the PMITS is not a memory-intensive algorithm, but we set the 8 GB of RAM memory limit for CPLEX. Solutions for BiQAP are not reported since no model could provide a feasible solution for any instance tested due to the huge memory required to model and solve this problem. For the remaining problems, the upper (UB) and lower (LB) bounds provided by CPLEX after a 1-hour run for each instance tested are shown in Tables C.1–C.4. Solutions highlighted are for the model that optimality is proved faster. In case of ties, we highlight the method with lower UB and, persisting the tie, the method with higher LB. Finally, "n.s." means that the corresponding model could not provide a feasible solution for the problem within the time and memory limit.

Table C.1 – Comparison of different programming models for the QAP

Instance	*BKS*	*QIP*			*Standard*			*RLT-1*		
		UB	*LB*	*Time (min)*	*UB*	*LB*	*Time (min)*	*UB*	*LB*	*Time (min)*
tai20a	703,482	**716,382**	0	> 60	878,790	618599.2	> 60	766,572	0	> 60
tai25a	1,167,256	1,259,462	0	> 60	1,431,716	0	> 60	**1,256,316**	0	> 60
tai30a	1,818,146	**1,956,590**	0	> 60	2,223,712	0	> 60	2,065,198	0	> 60
tai35a	2,422,002	**2,761,460**	0	> 60	2,896,892	0	> 60	2,896,892	0	> 60
tai40a	3,139,370	**3,578,260**	0	> 60	3,852,726	0	> 60	3,852,726	0	> 60
tai50a	4,938,796	**5,684,732**	0	> 60	5,941,988	0	> 60	5,941,988	0	> 60
tai60a	7,205,962	**8,330,288**	0	> 60	8,469,610	0	> 60	8,469,610	0	> 60
tai80a	13,499,184	**15,546,972**	0	> 60	15,688,718	0	> 60	15,688,718	0	> 60
tai100a	21,052,466		*n.s.*			*n.s.*			*n.s.*	
tai20b	122,455,319	**123,614,297**	0	> 60	283,943,307	0	> 60	133,399,543	0	> 60
tai25b	344,355,646	**346,483,036**	0	> 60	868,229,041	0	> 60	405,566,525	0	> 60
tai30b	637,117,113	**680,108,642**	0	> 60	1,354,588,506	0	> 60	822,850,107	0	> 60
tai35b	283,315,445	**322,097,099**	0	> 60	442,359,892	0	> 60	441,409,090	0	> 60
tai40b	637,250,948	**769,041,254**	0	> 60	1,204,324,820	0	> 60	1,204,324,820	0	> 60
tai50b	458,821,517	**688,807,919**	0	> 60	711,391,293	0	> 60	711,391,293	0	> 60
tai60b	608,215,054	**911,005,849**	0	> 60	1,027,374,245	0	> 60	1,027,374,245	0	> 60
tai80b	818,415,043	**1,259,942,652**	0	> 60		*n.s.*			*n.s.*	
tai100b	1,185,996,137		*n.s.*			*n.s.*			*n.s.*	
sko42	15,812	**18,454**	480	> 60	20,566	0	> 60	20,566	0	> 60
sko49	23,386	**27,112**	700	> 60	28,712	0	> 60	28,712	0	> 60
sko56	34,458	**40,694**	0	> 60	42,650	0	> 60	42,650	0	> 60
sko64	48,498	**58,330**	0	> 60	59,838	0	> 60	59,838	0	> 60
sko72	66,256	**78,830**	0	> 60	80,474	0	> 60	80,474	0	> 60
sko81	90,998	**107,062**	0	> 60	108,562	0	> 60	108,562	0	> 60
sko90	115,534	**134,998**	0	> 60	135,724	0	> 60	135,724	0	> 60
sko100a	152,002		*n.s.*			*n.s.*			*n.s.*	
sko100b	153,890		*n.s.*			*n.s.*			*n.s.*	
sko100c	147,862		*n.s.*			*n.s.*			*n.s.*	
sko100d	149,576		*n.s.*			*n.s.*			*n.s.*	
sko100e	149,150		*n.s.*			*n.s.*			*n.s.*	

Table C.2 – Comparison of different programming models for the QBAP

Instance	*BKS*	*QIP*			*Standard*			*RLT-1*		
		UB	*LB*	*Time (min)*	*UB*	*LB*	*Time (min)*	*UB*	*LB*	*Time (min)*
tai10a	4,256	4,256	4,256	1.0	4,256	4,256	5.6	**4,256**	4,256	0.1
tai12a	4,756	4,756	4,756	5.3	4,756	4,756	55.1	**4,756**	4,756	0.2
tai15a	4,757	4,757	4,757	57.3	7,917	21.3	> 60	**4,757**	4,757	0.9
tai17a	4,704	9,312	0	> 60	8,170	14.2	> 60	**4,704**	4,704	16.3
tai20a	5,096		*n.s.*		8,742	10.7	> 60	**5,200**	103.2	> 60
tai25a	5,328		*n.s.*		9,024	8.2	> 60	**5,880**	0	> 60
tai30a	5,952		*n.s.*		8,930	0	> 60	**6,992**	0	> 60
tai35a	6,120		*n.s.*		**8,613**	0	> 60	**8,613**	0	> 60
tai40a	6,370		*n.s.*		**9,306**	0	> 60	**9,306**	0	> 60
tai50a	6,873		*n.s.*		**9,604**	0	> 60	**9,604**	0	> 60
tai60a	7,134		*n.s.*		**9,702**	0	> 60	**9,702**	0	> 60
tai80a	7,695		*n.s.*			*n.s.*			*n.s.*	
tai100a	8,036		*n.s.*			*n.s.*			*n.s.*	
tai12b	4,371,380	4,371,380	4,371,380	1.7	19,230,438	30,190.4	> 60	**4,371,380**	4,371,380	0.6
tai15b	22,204,329	22,204,329	22,204,329	6.5	291,946,422	549,753.1	> 60	**22,204,329**	22,204,329	0.7
tai20b	17,132,916	36,470,412	0	> 60	45,979,548	28,405.9	> 60	**17,132,916**	0	> 60
tai25b	17,166,072		*n.s.*		52,942,159	0	> 60	**31,797,414**	0	> 60
tai30b	25,462,115		*n.s.*		62,752,438	0	> 60	**49,847,879**	0	> 60
tai35b	7,159,056		*n.s.*		**16,153,962**	0	> 60	**16,153,962**	0	> 60
tai40b	12,566,129		*n.s.*		**24,261,283**	0	> 60	**24,261,283**	0	> 60
tai50b	5,180,968		*n.s.*		**13,002,408**	0	> 60	**13,002,408**	0	> 60
tai60b	5,800,564		*n.s.*		**15,427,179**	0	> 60	**15,427,179**	0	> 60
tai80b	3,335,332		*n.s.*			*n.s.*			*n.s.*	
tai100b	4,188,992		*n.s.*			*n.s.*			*n.s.*	
sko42	50		*n.s.*		**100**	0	> 60	**100**	0	> 60
sko49	60		*n.s.*		**110**	0	> 60	**110**	0	> 60
sko56	70		*n.s.*		**120**	0	> 60	**120**	0	> 60
sko64	80		*n.s.*		**130**	0	> 60		*n.s.*	
sko72	90		*n.s.*		**140**	0	> 60		*n.s.*	
sko81	100		*n.s.*			*n.s.*			*n.s.*	
sko90	100		*n.s.*			*n.s.*			*n.s.*	
sko100a	110		*n.s.*			*n.s.*			*n.s.*	
sko100b	120		*n.s.*			*n.s.*			*n.s.*	
sko100c	110		*n.s.*			*n.s.*			*n.s.*	
sko100d	120		*n.s.*			*n.s.*			*n.s.*	
sko100e	120		*n.s.*			*n.s.*			*n.s.*	
sko100f	110		*n.s.*			*n.s.*			*n.s.*	

Table C.3 – Comparison of different programming models for the QSAP

Instance	*BKS*	*QIP*			*Standard*			*RLT-1*		
		UB	*LB*	*Time (min)*	*UB*	*LB*	*Time (min)*	*UB*	*LB*	*Time (min)*
20-15-35	1,599,473	1,599,473	0	> 60	**1,599,473**	1,599,473	0.4	1,743,672	950,356.9	> 60
20-15-55	1,427,052	1,427,052	0	> 60	**1,427,052**	1,427,052	0.2	1,427,052	1,227,895.5	> 60
20-15-75	1,648,679	1,657,225	0	> 60	**1,648,679**	1,648,679	0.4	1,688,867	1,200,323.1	> 60
30-06-95	5,486,902	5,486,902	4,105,696.8	> 60	**5,486,902**	5,486,902	0.1	5,594,528	4,491,337.9	> 60
30-07-75	4,834,397	4,834,397	2,144,853.4	> 60	**4,834,397**	4,834,397	0.1	4,903,530	3,886,765.5	> 60
30-08-55	4,484,813	4,484,813	1,622,616.9	> 60	**4,484,813**	4,484,813	0.2	4,807,529	3,076,850.8	> 60
30-10-65	3,649,165	3,649,165	0	> 60	**3,649,165**	3,649,165	0.2	3,941,471	2,659,682.7	> 60
30-20-35	3,351,755	3,548,616	1,445,131.9	> 60	**3,351,755**	3,351,755	15.0	4,164,734	1,389,583.5	> 60
30-20-55	3,247,260	3,626,523	1,467,746.9	> 60	**3,247,260**	3,247,260	6.6	4,219,973	1,367,307	> 60
30-20-75	3,301,384	3,891,689	1,323,822.8	> 60	**3,301,384**	3,128,684.2	> 60	3,953,122	1,299,154	> 60
30-20-95	2,941,907	3,363,980	1,438,963	> 60	**2,941,907**	2,941,907	3.0	3,939,205	1,370,752	> 60
35-15-35	4,533,539	5,029,510	2,349,754.3	> 60	**4,533,539**	4,533,539	3.2	5,493,652	2,095,453.5	> 60
35-15-55	4,220,924	4,693,131	2,288,238.8	> 60	**4,220,924**	4,220,924	2.0	5,289,805	2,196,135.5	> 60
35-15-75	5,620,789	6,285,897	2,233,543.4	> 60	**5,620,789**	5,387,465	> 60	6,984,087	2,173,483.5	> 60
35-15-95	4,555,240	4,907,006	2,192,591.2	> 60	**4,555,240**	4,555,240	5.7	5,287,916	2,046,658	> 60
40-07-75	8,347,601	8,347,601	4,903,659.1	> 60	**8,347,601**	8,347,601	0.8	8,737,467	5,694,638.2	> 60
40-09-95	7,107,977	7,107,977	0	> 60	**7,107,977**	7,107,977	1.9	8,133,094	3,910,355.4	> 60
40-10-65	7,509,269	7,509,269	0	> 60	**7,509,269**	7,509,269	2.2	8,377,608	3,642,050.2	> 60
50-10-65	11,795,583	12,492,942	5,797,763.7	> 60	**11,795,583**	11,795,583	2.7	12,992,679	5,014,669.8	> 60
50-10-75	10,107,391	11,319,400	5,358,212.8	> 60	**10,107,391**	10,107,391	1.5	12,402,134	4,754,003.3	> 60
50-10-95	11,882,812	13,603,227	5,843,897.8	> 60	**11,882,812**	11,882,812	15.6	14,560,121	4,527,029.3	> 60
50-C1	1,022,084	1,092,678	41,032.9	> 60	**1,022,084**	835,043.1	> 60	1,152,523	109,358	> 60
50-C10	1,123,607	1,215,531	122,053.5	> 60	**1,124,895**	881,573.8	> 60	1,276,380	116,293	> 60
50-C25	1,292,223	1,404,812	137,076.1	> 60	**1,293,354**	993,767.4	> 60	1,421,132	137,163	> 60
50-C50	1,573,474	1,649,234	173,004.7	> 60	**1,573,474**	1,184,582.1	> 60	1,717,190	177,208	> 60
75-C1	1,993,829	2,229,276	19,951	> 60	**2,045,813**	1,241,799.7	> 60	2,698,381	20,837.5	> 60
75-C10	2,195,019	2,517,200	34,183.5	> 60	**2,232,142**	1,347,948.3	> 60	2,618,208	33,113.5	> 60
75-C25	2,530,699	2,912,117	51,484.5	> 60	**2,581,945**	1,530,696.1	> 60	3,046,568	48,664	> 60
75-C50	3,089,736	3,632,103	67,276	> 60	**3,124,028**	1,838,668.9	> 60	3,544,038	67,955.8	> 60
100-C1	3,490,341	**3,731,562**	16,847	> 60	6,539,252	0	> 60	4,418,790	16,541	> 60
100-C10	3,836,299	**4,469,237**	33,257.5	> 60	7,122,728	0	> 60	5,096,063	33,099	> 60
100-C25	4,412,117	**5,085,661**	48,419	> 60	8,094,043	0	> 60	5,291,774	48,589	> 60
100-C50	5,370,205	**6,083,200**	66,634.5	> 60	9,712,705	0	> 60	6,304,163	68,208	> 60
125-C1	4,881,972	**5,358,958**	14,545.5	> 60	10,279,741	0	> 60	10,359,500	12,874.5	> 60
125-C10	5,375,051	**6,261,164**	32,730	> 60	11,194,743	0	> 60	11,272,100	29,013	> 60
125-C25	6,196,362	**6,713,149**	48,074	> 60	12,719,555	0	> 60	12,794,500	45,046	> 60
125-C50	7,564,714	**7,902,849**	68,468.5	> 60	15,260,785	0	> 60	15,330,000	63,558.3	> 60
150-C1	6,930,942	**14,631,247**	15,349	> 60	14,631,247	0	> 60	14,635,100	14,832.5	> 60
150-C10	7,625,357	**15,934,010**	34,671.5	> 60	15,934,010	0	> 60	15,939,600	34,596	> 60
150-C25	8,782,100	**18,104,789**	51,740.5	> 60	18,104,789	0	> 60	18,112,100	52,116.5	> 60
150-C50	10,707,271	**21,721,652**	71,162	> 60	21,721,652	0	> 60	21,733,200	71,387.3	> 60

Table C.4 – Comparison of different programming models for the GQAP

Instance	*BKS*	*QIP*			*Standard*			*RLT-1*		
		UB	*LB*	*Time (min)*	*UB*	*LB*	*Time (min)*	*UB*	*LB*	*Time (min)*
20-15-35	1,471,896	1,497,245	0	> 60	**1,471,896**	1,471,896	1.4	1,580,882	1,090,133	> 60
20-15-55	1,723,638	1,738,832	0	> 60	**1,723,638**	1,723,638	3.1	1,786,857	1,326,449.2	> 60
20-15-75	1,953,188	1,962,929	0	> 60	**1,953,188**	1,953,188	0.8	1,998,760	1,605,052	> 60
30-06-95	5,160,920	5,160,920	1,428,686.1	> 60	**5,160,920**	5,160,920	2.9	5,353,379	4,049,777.5	> 60
30-07-75	4,383,923	4,383,923	0	> 60	**4,383,923**	4,383,923	3.8	4,426,503	3,476,886	> 60
30-08-55	3,501,695	3,501,695	0	> 60	**3,501,695**	3,501,695	0.2	3,639,346	3,167,891.6	> 60
30-10-65	3,620,959	3,624,283	0	> 60	**3,620,959**	3,620,959	14.0	3,941,213	2,411,063.6	> 60
30-20-35	3,379,359	4,049,703	1,327,706.2	> 60	**3,456,071**	2,835,598.5	> 60	4,270,683	1,311,181.2	> 60
30-20-55	3,593,105	4,098,897	1,246,632.9	> 60	**3,666,263**	2,760,724.1	> 60	4,291,262	1,240,264.6	> 60
30-20-75	4,050,938	4,596,404	1,357,346.6	> 60	**4,104,358**	3,141,123.5	> 60	4,626,243	1,102,779.4	> 60
30-20-95	5,710,645	**5,726,530**	0	> 60	5,732,543	5,037,035.9	> 60	6,415,498	3,098,463	> 60
35-15-35	4,456,670	**4,456,670**	0	> 60	4,464,070	4,030,016.9	> 60	5,416,737	2,008,800.7	> 60
35-15-55	4,639,128	4,670,219	0	> 60	**4,639,128**	3,817,013.8	> 60	5,484,906	1,980,152.6	> 60
35-15-75	6,301,723	**6,301,723**	0	> 60	6,385,200	4,344,303.6	> 60	7,151,400	1,725,440.8	> 60
35-15-95	6,670,264	**6,670,264**	0	> 60	6,978,531	4,671,615.4	> 60	8,270,344	1,685,809.8	> 60
40-07-75	7,405,793	7,405,793	0	> 60	**7,405,793**	7,405,793	28.2	7,878,293	5,190,747.1	> 60
40-09-95	7,667,719	**7,762,421**	0	> 60	7,858,037	6,376,979.7	> 60	8,528,686	3,578,848.3	> 60
40-10-65	7,265,559	7,265,559	0	> 60	**7,265,559**	6,669,482.3	> 60	8,065,464	3,510,270.4	> 60
50-10-65	10,513,029	10,513,029	0	> 60	**10,513,029**	10,212,775.2	> 60	12,348,522	5,118,268.3	> 60
50-10-75	11,217,503	**11,251,072**	0	> 60	11,407,879	9,179,025.3	> 60	13,063,993	4,480,847	> 60
50-10-95	12,845,598	**12,877,686**	0	> 60	13,172,803	9,816,920	> 60	14,301,474	5,009,641.2	> 60

Bibliography

M. Abdel-Basset, G. Manogaran, D. El-Shahat, and S. Mirjalili. Integrating the whale algorithm with tabu search for quadratic assignment problem: a new approach for locating hospital departments. *Applied soft computing*, 73:530–546, 2018a.

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

O. Abdelkafi, L. Idoumghar, and J. Lepagnot. A survey on the metaheuristics applied to QAP for the graphics processing units. *Parallel Processing Letters*, 26(03):1650013, 2016.

O. Abdelkafi, B. Derbel, and A. Liefooghe. A parallel tabu search for the large-scale quadratic assignment problem. In *2019 IEEE Congress on Evolutionary Computation (CEC)*, pages 3070–3077. IEEE, 2019.

W. P. Adams, M. Guignard, P. M. Hahn, and W. L. Hightower. A level-2 reformulation–linearization technique bound for the quadratic assignment problem. *European Journal of Operational Research*, 180(3):983–996, 2007.

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*, 129:105168, 2021.

C. C. Aggarwal. *Neural Networks and Deep Learning*. Springer, Berlin, 2018.

Y. Aksan, T. Dokeroglu, and A. Cosar. A stagnation-aware cooperative parallel breakout local search algorithm for the quadratic assignment problem. *Computers & Industrial Engineering*, 103:105–115, 2017.

F. J. Aldarondo and Y. A. Bozer. Expected distances and alternative design configurations for automated guided vehicle-based order picking systems. *International Journal of Production Research*, pages 1–18, 2020.

D. L. Applegate, R. E. Bixby, V. Chvatal, and W. J. Cook. *The Traveling Salesman Problem: A Computational Study*. Princeton University Press, Princeton, 2007.

K. Azadeh, R. De Koster, and D. Roy. Robotized and automated warehouse systems: Review and recent developments. *Transportation Science*, 53(4):917–945, 2019.

D. Battini, M. Calzavara, A. Persona, and F. Sgarbossa. Order picking system design: the storage assignment and travel distance estimation (SA&TDE) joint method. *International Journal of Production Research*, 53(4):1077–1093, 2015.

S. Benjaafar. Modeling and analysis of congestion in the design of facility layouts. *Management Science*, 48(5):679–704, 2002.

U. Benlic and J.-K. Hao. Breakout local search for the quadratic assignment problem. *Applied Mathematics and Computation*, 219(9):4800–4815, 2013.

U. Benlic and J.-K. Hao. Memetic search for the quadratic assignment problem. *Expert Systems with Applications*, 42(1):584–595, 2015.

A. Billionnet and S. Elloumi. Best reduction of the quadratic semi-assignment problem. *Discrete Applied Mathematics*, 109(3):197–213, 2001.

A. Billionnet, M.-C. Costa, and A. Sutter. An efficient algorithm for a task allocation problem. *Journal of the Association for Computing Machinery*, 39(3):502–518, 1992.

P. Bodnar and J. Lysgaard. A dynamic programming algorithm for the space allocation and aisle positioning problem. *Journal of the Operational Research Society*, 65(9):1315–1324, 2014.

M. Bortolini, E. Bottani, and H. Eric. Green warehousing: Systematic literature review and bibliometric analysis. *Journal of Cleaner Production*, 2019a.

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*, 100(9-12):2521–2536, 2019b.

J. Bos. Zoning in forest management: A quadratic assignment problem solved by simulated annealing. *Journal of Environmental Management*, 37(2):127–145, 1993.

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

N. Boysen, D. Briskorn, and S. Emde. Parts-to-picker based order processing in a rack-moving mobile robots environment. *European Journal of Operational Research*, 262(2):550–562, 2017.

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

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

L. Breiman. Random forests. *Machine Learning*, 45(1):5–32, 2001.

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*, 285(2):497–512, 2020.

R. E. Burkard. Selected topics on assignment problems. *Discrete Applied Mathematics*, 123 (1-3):257–302, 2002.

R. E. Burkard. Quadratic assignment problems. In P. M. Pardalos, D.-Z. Du, and R. L. Graham, editors, *Handbook of Combinatorial Optimization*, pages 2741–2814. Springer, London, 2013.

R. E. Burkard and E. Çela. Heuristics for biquadratic assignment problems and their computational comparison. *European Journal of Operational Research*, 83(2):283–300, 1995.

R. E. Burkard, E. Çela, and B. Klinz. On the biquadratic assignment problem. In P. M. Pardalos and H. Wolkowicz, editors, *Quadratic Assignment and Related Problems, DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, volume 16, pages 117–146. American Mathematical Society, 1994.

R. E. Burkard, S. E. Karisch, and F. Rendl. QAPLIB – a quadratic assignment problem library. *Journal of Global Optimization*, 10(4):391–403, 1997.

M. Calzavara, F. Sgarbossa, and A. Persona. Vertical lift modules for small items order picking: an economic evaluation. *International Journal of Production Economics*, 210:199–210, 2019.

H. J. Carlo and G. E. Giraldo. Toward perpetually organized unit-load warehouses. *Computers & Industrial Engineering*, 63(4):1003–1012, 2012.

F. Caron, G. Marchet, and A. Perego. Routing policies and COI-based storage policies in picker-to-part systems. *International journal of production research*, 36(3):713–732, 1998.

E. Çela, V. Deineko, and G. J. Woeginger. New special cases of the quadratic assignment problem with diagonally structured coefficient matrices. *European Journal of Operational Research*, 267(3):818–834, 2018.

M. Çelik and H. Süral. Order picking in parallel-aisle warehouses with multiple blocks: Complexity and a graph theory-based heuristic. *International Journal of Production Research*, 57(3):888–906, 2019.

E. Çeven and K. R. Gue. Optimal wave release times for order fulfillment systems with deadlines. *Transportation Science*, 51(1):52–66, 2017.

T. Chabot, R. Lahyani, L. C. Coelho, and J. Renaud. Order picking problems under weight, fragility and category constraints. *International Journal of Production Research*, 55(21): 6361–6379, 2017.

T. Chabot, L. C. Coelho, J. Renaud, and J.-F. Côté. Mathematical model, heuristics and exact method for order picking in narrow aisles. *Journal of the Operational Research Society*, 69 (8):1242–1253, 2018.

F. T. S. Chan and H. K. Chan. Improving the productivity of order picking of a manual-pick and multi-level rack distribution warehouse through the implementation of class-based storage. *Expert Systems with Applications*, 38(3):2686–2700, 2011.

C.-M. Chen, Y. Gong, R. B. M. De Koster, and J. A. E. E. Van Nunen. A flexible evaluative framework for order picking systems. *Production and Operations Management*, 19(1):70–82, 2010.

L. Chen, A. Langevin, and D. Riopel. A tabu search algorithm for the relocation problem in a warehousing system. *International Journal of Production Economics*, 129(1):147–156, 2011.

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

W.-C. Chiang and C. Chiang. Intelligent local search strategies for solving facility layout problems with the quadratic assignment problem formulation. *European Journal of Operational Research*, 106(2-3):457–488, 1998.

W.-C. Chiang, P. Kouvelis, and T. L. Urban. Incorporating workflow interference in facility layout design: the quartic assignment problem. *Management Science*, 48(4):584–590, 2002.

N. Christofides and I. Colloff. The rearrangement of items in a warehouse. *Operations Research*, 21(2):577–589, 1973.

Y.-F. Chuang, H.-T. Lee, and Y.-C. Lai. Item-associated cluster assignment model on storage allocation problems. *Computers & Industrial Engineering*, 63(4):1171–1177, 2012.

J. Clausen and M. Perregaard. Solving large quadratic assignment problems in parallel. *Computational Optimization and Applications*, 8(2):111–127, 1997.

J.-F. Cordeau, M. Gaudioso, G. Laporte, and L. Moccia. A memetic heuristic for the generalized quadratic assignment problem. *INFORMS Journal on Computing*, 18(4):433–443, 2006.

Cplusplus. Map, 2020. http://www.cplusplus.com/reference/map/map/.

M. Czapiński. An effective parallel multistart tabu search for quadratic assignment problem on CUDA platform. *Journal of Parallel and Distributed Computing*, 73(11):1461–1468, 2013.

Í. R. da Costa Barros and T. P. do Nascimento. Robotic mobile fulfillment systems: A survey on recent developments and research opportunities. *Robotics and Autonomous Systems*, page 103729, 2021.

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

R. B. M. De Koster, T. Le-Duc, and N. Zaerpour. Determining the number of zones in a pick-and-sort order picking system. *International Journal of Production Research*, 50(3): 757–771, 2012.

R. Dekker, R. B. M. De Koster, K. J. Roodbergen, and H. Van Kalleveen. Improving order-picking response time at Ankor's warehouse. *Interfaces*, 34(4):303–313, 2004.

M. Dell'Amico, J. C. D. Diaz, M. Iori, and R. Montanari. The single-finger keyboard layout problem. *Computers & Operations Research*, 36(11):3002–3012, 2009.

J. W. Dickey and J. W. Hopkins. Campus building arrangement using TOPAZ. *Transportation Research*, 6(1):59–68, 1972.

A. S. Dijkstra and K. J. Roodbergen. Exact route-length formulas and a storage location assignment heuristic for picker-to-parts warehouses. *Transportation Research Part E: Logistics and Transportation Review*, 102:38–59, 2017.

T. Dokeroglu. Hybrid teaching–learning-based optimization algorithms for the quadratic assignment problem. *Computers & Industrial Engineering*, 85:86–101, 2015.

T. Dokeroglu and A. Cosar. A novel multistart hyper-heuristic algorithm on the grid for the quadratic assignment problem. *Engineering Applications of Artificial Intelligence*, 52:10–25, 2016.

T. Dokeroglu, E. Sevinc, and A. Cosar. Artificial bee colony optimization for the quadratic assignment problem. *Applied Soft Computing*, 76:595–606, 2019.

W. Domschke. Schedule synchronization for public transit networks. *Operations-Research-Spektrum*, 11(1):17–24, 1989.

W. Domschke, P. Forst, and S. Voß. Tabu search techniques for the quadratic semi-assignment problem. In G. Fandel, T. Gulledge, and A. Jones, editors, *New Directions for Operations Research in Manufacturing*, pages 389–405. Springer, Berlin, 1992.

Z. Drezner. A new genetic algorithm for the quadratic assignment problem. *INFORMS Journal on Computing*, 15(3):320–330, 2003.

Z. Drezner. Compounded genetic algorithms for the quadratic assignment problem. *Operations Research Letters*, 33(5):475–480, 2005a.

Z. Drezner. The extended concentric tabu for the quadratic assignment problem. *European Journal of Operational Research*, 160(2):416–422, 2005b.

Z. Drezner. Extensive experiments with hybrid genetic algorithms for the solution of the quadratic assignment problem. *Computers & Operations Research*, 35(3):717–736, 2008.

M. Drwal. Algorithm for quadratic semi-assignment problem with partition size coefficients. *Optimization Letters*, 8(3):1183–1190, 2014.

E. Duman, M. Uysal, and A. F. Alkaya. Migrating Birds Optimization: A new metaheuristic approach and its performance on quadratic assignment problem. *Information Sciences*, 217: 65–77, 2012.

H. A. Eiselt and G. Laporte. A combinatorial optimization problem arising in dartboard design. *Journal of the Operational Research Society*, 42(2):113–118, 1991.

A. N. Elshafei. Hospital layout as a quadratic assignment problem. *Journal of the Operational Research Society*, 28(1):167–179, 1977.

J. J. Enright and P. R. Wurman. Optimization and coordinated autonomy in mobile fulfillment systems. In *Workshops at the twenty-fifth AAAI conference on artificial intelligence*, 2011.

F. Esposito, D. Malerba, G. Semeraro, and J. Kay. A comparative analysis of methods for pruning decision trees. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 19(5):476–491, 1997.

A. Eynan and M. J. Rosenblatt. Establishing zones in single-command class-based rectangular AS/RS. *IIE Transactions*, 26(1):38–46, 1994.

G. Finke, R. E. Burkard, and F. Rendl. Quadratic assignment problems. *Annals of Discrete Mathematics*, 31(1):61–82, 1987.

M. Fischetti, M. Monaci, and D. Salvagnin. Three ideas for the quadratic assignment problem. *Operations Research*, 60(4):954–964, 2012.

C. Fleurent and J. A. Ferland. Genetic hybrids for the quadratic assignment problem. In P. M. Pardalos and H. Wolkowicz, editors, *Quadratic Assignment and Related Problems, DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, volume 16, pages 173–187. American Mathematical Society, 1994.

R. J. Forrester. Tightening concise linear reformulations of 0-1 cubic programs. *Optimization*, 65(4):877–903, 2016.

G. Fragapane, R. B. M. De Koster, F. Sgarbossa, and J. O. Strandhagen. Planning and control of autonomous mobile robots for intralogistics: Literature review and research agenda. *European Journal of Operational Research*, 2021.

E. Frazelle. *World-Class Warehousing and Material Handling*. McGraw-Hill, New York, 2nd edition, 2016.

E. A. Frazelle and G. P. Sharp. Correlated assignment strategy can improve any order-picking operation. *Industrial Engineering*, 21(4):33–37, 1989.

A. M. Frieze and J. Yadegar. On the quadratic assignment problem. *Discrete Applied Mathematics*, 5(1):89–98, 1983.

M. C. Fu and B. K. Kaku. Minimizing work-in-process and material handling in the facilities layout problem. *IIE Transactions*, 29(1):29–36, 1997.

N. Gademann and S. Velde. Order batching to minimize total travel time in a parallel-aisle warehouse. *IIE transactions*, 37(1):63–75, 2005.

L. M. Gambardella, E. D. Taillard, and M. Dorigo. Ant colonies for the QAP. *Journal of Operational Research Society*, 50(2):167–176, 1999.

R. Garnier. Concurrent neural network: a model of competition between times series. *Annals of Operations Research*, pages 1–20, 2021.

A. Gharehgozli and N. Zaerpour. Robot scheduling for pod retrieval in a robotic mobile fulfillment system. *Transportation Research Part E: Logistics and Transportation Review*, 142:102087, 2020.

C. H. Glock, E. H. Grosse, W. P. Neumann, and A. Feldman. Assistive devices for manual materials handling in warehouses: a systematic literature review. *International Journal of Production Research*, pages 1–24, 2020.

F. Glover and E. Woolsey. Converting the 0–1 polynomial programming problem to a 0–1 linear program. *Operations Research*, 22(1):180–182, 1974.

Y. Gong, M. Jin, and Z. Yuan. Robotic mobile fulfilment systems considering customer classes. *International Journal of Production Research*, pages 1–18, 2020.

H. Greenberg. A quadratic assignment problem without column constraints. *Naval Research Logistics Quarterly*, 16(3):417–421, 1969.

E. H. Grosse, C. H. Glock, M. Y. Jaber, and W. P. Neumann. Incorporating human factors in order picking planning models: framework and research opportunities. *International Journal of Production Research*, 53(3):695–717, 2015.

E. H. Grosse, S. M. Dixon, W. P. Neumann, and C. H. Glock. Using qualitative interviewing to examine human factors in warehouse order picking. *International Journal of Logistics Systems and Management*, 23(4):499–518, 2016.

J. Gu, M. Goetschalckx, and L. F. McGinnis. Research on warehouse operation: A comprehensive review. *European Journal of Operational Research*, 177(1):1–21, 2007.

J. Gu, M. Goetschalckx, and L. F. McGinnis. Research on warehouse design and performance evaluation: A comprehensive review. *European Journal of Operational Research*, 203(3): 539–549, 2010.

M. Guan and Z. Li. Genetic algorithm for scattered storage assignment in Kiva mobile fulfillment system. *American Journal of Operations Research*, 8(6):474–485, 2018.

K. R. Gue and R. D. Meller. Aisle configurations for unit-load warehouses. *IIE Transactions*, 41(3):171–182, 2009.

E. Guizzo. Three engineers, hundreds of robots, one warehouse. *IEEE Spectrum*, 45(7):26–34, 2008.

X. Guo, Y. Yu, and R. B. M. De Koster. Impact of required storage space on storage policy performance in a unit-load warehouse. *International Journal of Production Research*, 54(8): 2405–2418, 2016.

P. Hahn, J. M. Smith, and Y.-R. Zhu. The multi-story space assignment problem. *Annals of Operations Research*, 179(1):77–103, 2010.

P. M. Hahn, B.-J. Kim, M. Guignard, J. M. Smith, and Y.-R. Zhu. An algorithm for the generalized quadratic assignment problem. *Computational Optimization and Applications*, 40(3):351, 2008a.

P. M. Hahn, B.-J. Kim, T. Stützle, S. Kanthak, W. L. Hightower, H. Samra, Z. Ding, and M. Guignard. The quadratic three-dimensional assignment problem: Exact and approximate solution methods. *European Journal of Operational Research*, 184(2):416–428, 2008b.

P. M. Hahn, Y.-R. Zhu, M. Guignard, W. L. Hightower, and M. J. Saltzman. A level-3 reformulation-linearization technique-based bound for the quadratic assignment problem. *INFORMS Journal on Computing*, 24(2):202–209, 2012.

P. Hansen and K.-W. Lih. Improved algorithms for partitioning problems in parallel, pipelined, and distributed computing. *IEEE Transactions on Computers*, 41(6):769–771, 1992.

P. Hansen, N. Mladenović, J. Brimberg, and J. A. M. Pérez. Variable neighborhood search. In M. Gendreau and J.-Y. Potvin, editors, *Handbook of Metaheuristics*, pages 57–97. Springer, New York, 2019.

M. Harris, R. Berretta, M. Inostroza-Ponta, and P. Moscato. A memetic algorithm for the quadratic assignment problem with parallel local search. In *IEEE Congress on Evolutionary Computation (CEC)*, pages 838–845, 2015.

W. H. Hausman, L. B. Schwarz, and S. C. Graves. Optimal storage assignment in automatic warehousing systems. *Management Science*, 22(6):629–638, 1976.

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

S. Henn and V. Schmid. Metaheuristics for order batching and sequencing in manual order picking systems. *Computers & Industrial Engineering*, 66(2):338–351, 2013.

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

S. Henn, S. Koch, and G. Wäscher. Order batching in order picking warehouses: A survey of solution approaches. In R. Manzini, editor, *Warehousing in the Global Supply Chain: Advanced Models, Tools and Applications for Storage Systems*, pages 105–137. Springer, 2012.

J. L. Heskett. Cube-per-order index – a key to warehouse stock location. *Transportation and Distribution Management*, 3(1):27–31, 1963.

Y.-C. Ho and Y.-Y. Tseng. A study on order-batching methods of order-picking in a distribution centre with two cross-aisles. *International Journal of Production Research*, 44(17): 3391–3417, 2006.

J. N. Hooker and G. Ottosson. Logic-based Benders decomposition. *Mathematical Programming*, 96(1):33–60, 2003.

L.-F. Hsieh and Y.-C. Huang. New batch construction heuristics to optimise the performance of order picking systems. *International Journal of Production Economics*, 131(2):618–630, 2011.

L.-F. Hsieh and L. Tsai. The optimum design of a warehouse system on order picking efficiency. *The International Journal of Advanced Manufacturing Technology*, 28(5-6):626–637, 2006.

H. Hwang, Y. H. Oh, and Y. K. Lee. An evaluation of routing policies for order-picking operations in low-level picker-to-part system. *International Journal of Production Research*, 42(18):3873–3889, 2004.

Y. Jaghbeer, R. Hanson, and M. I. Johansson. Automated order picking systems and the links between design and performance: A systematic literature review. *International Journal of Production Research*, 58(15):4489–4505, 2020.

T. James, C. Rego, and F. Glover. Sequential and parallel path-relinking algorithms for the quadratic assignment problem. *IEEE Intelligent Systems*, 20(4):58–65, 2005.

T. James, C. Rego, and F. Glover. A cooperative parallel tabu search algorithm for the quadratic assignment problem. *European Journal of Operational Research*, 195(3):810–826, 2009a.

T. James, C. Rego, and F. Glover. Multistart tabu search and diversification strategies for the quadratic assignment problem. *IEEE Transactions on Systems, Man, and Cybernetics – Part A: Systems and Humans*, 39(3):579–596, 2009b.

C.-C. Jane and Y.-W. Laih. A clustering algorithm for item assignment in a synchronized zone order picking system. *European Journal of Operational Research*, 166(2):489–496, 2005.

J. M. Jarvis and E. D. McDowell. Optimal product layout in an order picking warehouse. *IIE transactions*, 23(1):93–102, 1991.

M. Jiang, K. H. Leung, Z. Lyu, and G. Q. Huang. Picking-replenishment synchronization for robotic forward-reserve warehouses. *Transportation Research Part E: Logistics and Transportation Review*, 144:102138, 2020.

G. Jin, P. Yang, and G. Duan. Multiple deep layout of robotic mobile fulfillment system. In *2020 IEEE 7th International Conference on Industrial Engineering and Applications (ICIEA)*, pages 230–234. IEEE, 2020.

B. S. Kim and J. S. Smith. Slotting methodology using correlated improvement for a zone-based carton picking distribution system. *Computers & Industrial Engineering*, 62(1):286–295, 2012.

J. Kleinberg and E. Tardos. Approximation algorithms for classification problems with pairwise relationships: Metric labeling and Markov random fields. *Journal of the Association for Computing Machinery*, 49(5):616–639, 2002.

A. J. Kleywegt, A. Shapiro, and T. Homem-de-Mello. The sample average approximation method for stochastic discrete optimization. *SIAM Journal on Optimization*, 12(2):479–502, 2002.

M. Klodawski, R. Jachimowski, I. Jacyna-Golda, and M. Izdebski. Simulation analysis of order picking efficiency with congestion situations. *International Journal of Simulation Modelling*, 17(3):431–443, 2018.

J. Knowles and D. Corne. Instance generators and test suites for the multiobjective quadratic assignment problem. In C. M. Fonseca, P. J. Fleming, E. Zitzler, K. Deb, and L. Thiele, editors, *Proceedings of the Evolutionary Multi-Criterion Optimization (EMO 2003), Second International Conference*, pages 295–310, 2003.

M. Kofler, A. Beham, S. Wagner, M. Affenzeller, and W. Achleitner. Re-warehousing vs. healing: Strategies for warehouse storage location assignment. In *Logistics and Industrial Informatics (LINDI), 2011 3rd IEEE International Symposium on*, pages 77–82. IEEE, 2011.

M. Kofler, A. Beham, S. Wagner, and M. Affenzeller. Affinity based slotting in warehouses with dynamic order patterns. In *Advanced Methods and Applications in Computational Intelligence*, pages 123–143. Springer, 2014.

T. C. Koopmans and M. Beckmann. Assignment problems and the location of economic activities. *Econometrica: Journal of the Econometric Society*, pages 53–76, 1957.

J. Krarup and P. M. Pruzan. Computer-aided layout design. In M. L. Balinski and C. Lemarechal, editors, *Mathematical Programming in Use*, pages 75–94. Springer, Berlin, 1978.

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*, 147:106645, 2020.

N. V. Kumar and C. S. Kumar. Development of collision free path planning algorithm for warehouse mobile robot. *Procedia Computer Science*, 133:456–463, 2018.

T. Lamballais, D. Roy, and R. B. M. De Koster. Estimating performance in a robotic mobile fulfillment system. *European Journal of Operational Research*, 256(3):976–990, 2017.

T. Lamballais, D. Roy, and R. B. M. De Koster. Inventory allocation in robotic mobile fulfillment systems. *IISE Transactions*, 52(1):1–17, 2020.

T. N. Larson, H. March, and A. Kusiak. A heuristic approach to warehouse layout with class-based storage. *IIE Transactions*, 29(4):337–348, 1997.

E. L. Lawler. The quadratic assignment problem. *Management Science*, 9(4):586–599, 1963.

E. L. Lawler. The quadratic assignment problem: a brief review. In B. Roy, editor, *Combinatorial Programming: Methods and Applications, NATO Advanced Study Institutes Series (Series C – Mathematical and Physical Sciences)*, volume 19, pages 351–360, 1975.

T. Le-Duc and R. B. M. De Koster. Travel distance estimation and storage zone optimization in a 2-block class-based storage strategy warehouse. *International Journal of Production Research*, 43(17):3561–3581, 2005.

C.-G. Lee and Z. Ma. The generalized quadratic assignment problem. Technical Report MIE-OR TR2005-01, Departament of Mechanical and Industrial Engineering, University of Toronto, Toronto, 2004.

M.-K. Lee and E. A. Elsayed. Optimization of warehouse storage capacity under a dedicated storage policy. *International Journal of Production Research*, 43(9):1785–1805, 2005.

J. Li, M. Moghaddam, and S. Y. Nof. Dynamic storage assignment with product affinity and ABC classification – a case study. *The International Journal of Advanced Manufacturing Technology*, 84(9-12):2179–2194, 2016.

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*, 55(2): 447–461, 2017.

J.-T Li and H.-J. Liu. Design optimization of Amazon robotics. *Automation, Control and Intelligent Systems*, 4(2):48, 2016.

X. Li, G. Hua, A. Huang, J.-B. Sheu, T. C. E. Cheng, and F. Huang. Storage assignment policy with awareness of energy consumption in the Kiva mobile fulfilment system. *Transportation Research Part E: Logistics and Transportation Review*, 144:102158, 2020.

C.-C. Lin, J.-R. Kang, C.-C. Hou, and C.-Y. Cheng. Joint order batching and picker Manhattan routing problem. *Computers & Industrial Engineering*, 95:164–174, 2016.

W.-Y. Loh. Fifty years of classification and regression trees. *International Statistical Review*, 82(3):329–348, 2014.

E. M. Loiola, N. M. M. de Abreu, P. O. Boaventura-Netto, P. Hahn, and T. Querido. A survey for the quadratic assignment problem. *European Journal of Operational Research*, 176(2): 657–690, 2007.

J. López, D. Múnera, D. Diaz, and S. Abreu. On integrating population-based metaheuristics with cooperative parallelism. In *2018 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*, pages 601–608, Vancouver, 2018a. IEEE.

J. López, D. Múnera, D. Diaz, and S. Abreu. Weaving of metaheuristics with cooperative parallelism. In A. Auger, C. M. Fonseca, N. Lourenço, P. Machado, L. Paquete, and D. Whitley, editors, *International Conference on Parallel Problem Solving from Nature*, pages 436–448, Coimbra, 2018b. Springer.

Y. Ma, Z. Zhang, A. Ihler, and B. Pan. Estimating warehouse rental price using machine learning techniques. *International Journal of Computers, Communications & Control*, 13 (2), 2018.

V. F. Magirou and J. Z. Milis. An algorithm for the multiprocessor assignment problem. *Operations Research Letters*, 8(6):351–356, 1989.

P. A. Makris, A. P. Makri, and C. G. Provatidis. Energy-saving methodology for material handling applications. *Applied Energy*, 83(10):1116–1124, 2006.

F. Malucelli. A polynomially solvable class of quadratic semi-assignment problems. *European Journal of Operational Research*, 91(3):619–622, 1996.

F. Malucelli and D. Pretolani. Lower bounds for the quadratic semi-assignment problem. *European Journal of Operational Research*, 83(2):365–375, 1995.

R. J. Mantel, P. C. Schuur, and S. S. Heragu. Order oriented slotting: A new assignment strategy for warehouses. *European Journal of Industrial Engineering*, 1(3):301–316, 2007.

R. Manzini, R. Accorsi, M. Gamberi, and S. Penazzi. Modeling class-based storage assignment over life cycle picking patterns. *International Journal of Production Economics*, 170:790–800, 2015.

T. Marill and D. Green. On the effectiveness of receptors in recognition systems. *IEEE Transactions on Information Theory*, 9(1):11–17, 1963.

G. R. Mateus, M. G. C. Resende, and R. M. A. Silva. GRASP with path-relinking for the generalized quadratic assignment problem. *Journal of Heuristics*, 17(5):527–565, 2011.

T. Mavridou, P. M. Pardalos, L. S. Pitsoulis, and M. G. C. Resende. A GRASP for the biquadratic assignment problem. *European Journal of Operational Research*, 105(3):613–621, 1998.

A. McKendall and C. Li. A tabu search heuristic for a generalized quadratic assignment problem. *Journal of Industrial and Production Engineering*, 34(3):221–231, 2017.

M. Merschformann, L. Xie, and H. Li. Rawsim-o: A simulation framework for robotic mobile fulfillment systems. *Logistics Research*, 11(8):1–11, 2018.

M. Merschformann, T. Lamballais, R. De Koster, and L. Suhl. Decision rules for robotic mobile fulfillment systems. *Operations Research Perspectives*, 6:100128, 2019.

K. Mihić, K. Ryan, and A. Wood. Randomized decomposition solver with the quadratic assignment problem as a case study. *INFORMS Journal on Computing*, 30(2):295–308, 2018.

I. Z. Milis and V. F. Magirou. A lagrangian relaxation algorithm for sparse quadratic assignment problems. *Operations Research Letters*, 17(2):69–76, 1995.

D. Ming-Huang Chiang, C.-P. Lin, and M.-C. Chen. Data mining based storage assignment heuristics for travel distance reduction. *Expert Systems*, 31(1):81–90, 2014.

G. Miranda, H. P. L. Luna, G. R. Mateus, and R. P. M. Ferreira. A performance guarantee heuristic for electronic components placement problems including thermal effects. *Computers & Operations Research*, 32(11):2937–2957, 2005.

D. Mirčetić, N. Ralević, S. Nikoličić, M. Maslarić, and D. Stojanović. Expert system models for forecasting forklifts engagement in a warehouse loading operation: A case study. *Promet – Traffic&Transportation*, 28(4):393–401, 2016.

M. Mirzaei, N. Zaerpour, and R. B. M. De Koster. The impact of integrated cluster-based storage allocation on parts-to-picker warehouse performance. *Transportation Research Part E: Logistics and Transportation Review*, 146:102207, 2021.

A. Misevičius. A modified simulated annealing algorithm for the quadratic assignment problem. *Informatica*, 14(4):497–514, 2003.

A. Misevičius. An improved hybrid optimization algorithm for the quadratic assignment problem. *Mathematical Modelling and Analysis*, 9(2):149–168, 2004.

A. Misevičius. An implementation of the iterated tabu search algorithm for the quadratic assignment problem. *OR Spectrum*, 34(3):665–690, 2012.

A. Misevičius. New best known solution for the most difficult QAP instance "tai100a". *Memetic Computing*, 11(3):331–332, 2019.

A. Misevičius and B. Kilda. Comparison of crossover operators for the quadratic assignment problem. *Information Technology and Control*, 34(2), 2005.

N. Mladenović and P. Hansen. Variable neighborhood search. *Computers & Operations Research*, 24(11):1097–1100, 1997.

M. C. Mountz, R. D'andrea, J. A. Laplante, P. L. I. David, P. K. Mansfield, and B. W. Amsbury. Inventory system with mobile drive unit and inventory holder, 2008. 7(402), 018.

C. H. Mowrey and P. J. Parikh. Mixed-width aisle configurations for order picking in distribution centers. *European Journal of Operational Research*, 232(1):87–97, 2014.

D. Munera, D. Diaz, and S. Abreu. Hybridization as cooperative parallelism for the quadratic assignment problem. In *International Workshop on Hybrid Metaheuristics*, pages 47–61. Springer, 2016.

V. R. Muppani and G. K. Adil. Efficient formation of storage classes for warehouse storage location assignment: A simulated annealing approach. *Omega*, 36(4):609–618, 2008.

J. M. R. Murteira and J. J. S. Ramalho. Regression analysis of multivariate fractional data. *Econometric Reviews*, 35(4):515–552, 2016.

G. Nagy and S. Salhi. Location-routing: Issues, models and methods. *European Journal of Operational Research*, 177(2):649–672, 2007.

D. Nguyen Duc, T. Tran Huu, and N. Nananukul. A dynamic route-planning system based on industry 4.0 technology. *Algorithms*, 13(12):308, 2020.

K. I. Nikolopoulos, M. Z. Babai, and K. Bozos. Forecasting supply chain sporadic demand with nearest neighbor approaches. *International Journal of Production Economics*, 177: 139–148, 2016.

C. Novoa, A. Qasem, and A. Chaparala. A SIMD tabu search implementation for solving the quadratic assignment problem with GPU acceleration. In *Proceedings of the 2015 XSEDE Conference: Scientific Advancements Enabled by Enhanced Cyberinfrastructure*, pages 1–8, 2015.

A. Nyberg and T. Westerlund. A new exact discrete linear reformulation of the quadratic assignment problem. *European Journal of Operational Research*, 220(2):314–319, 2012.

S. G. Ozden, A. E. Smith, and K. R. Gue. A novel approach for modeling order picking paths. *Naval Research Logistics*, 2020.

J. C.-H. Pan and M.-H. Wu. Throughput analysis for order picking system with multiple pickers and aisle congestion considerations. *Computers & Operations Research*, 39(7):1661–1672, 2012.

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

A. V. Panyukov and R. E. Shangin. Algorithm for the discrete Weber's problem with an accuracy estimate. *Automation and Remote Control*, 77(7):1208–1215, 2016.

P. M. Pardalos and J. V. Crouse. A parallel algorithm for the quadratic assignment problem. In *Proceedings of the Supercomputing Conference 1989*, pages 351–360. IEEE, ACM, 1989.

P. M. Pardalos, F. Rendl, and H. Wolkowicz. The quadratic assignment problem: A survey and recent developments. In P. M. Pardalos and H. Wolkowicz, editors, *Quadratic Assignment and Related Problems, DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, volume 16, pages 1–42. American Mathematical Society, 1994.

P. J. Parikh and R. D. Meller. A travel-time model for a person-onboard order picking system. *European Journal of Operational Research*, 200(2):385–394, 2010.

J. A Pazour and H. J. Carlo. Warehouse reshuffling: Insights and optimization. *Transportation Research Part E: Logistics and Transportation Review*, 73:207–226, 2015.

F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *The Journal of Machine Learning Research*, 12:2825–2830, 2011.

A. A. Pessoa, P. M. Hahn, M. Guignard, and Y.-R. Zhu. Algorithms for the generalized quadratic assignment problem combining Lagrangean decomposition and the Reformulation-Linearization Technique. *European Journal of Operational Research*, 206(1):54–63, 2010.

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

C. G. Petersen. The impact of routing and storage policies on warehouse efficiency. *International Journal of Operations & Production Management*, 19(10):1053–1064, 1999.

C. G. Petersen and G. R. Aase. A comparison of picking, storage, and routing policies in manual order picking. *International Journal of Production Economics*, 92(1):11–19, 2004.

C. G. Petersen, G. R. Aase, and D. R. Heiser. Improving order-picking performance through the implementation of class-based storage. *International Journal of Physical Distribution & Logistics Management*, 34(7):534–544, 2004.

A. T. Phillips and J. B. Rosen. A quadratic assignment formulation of the molecular conformation problem. *Journal of Global Optimization*, 4(2):229–241, 1994.

M. A. Pollatschek, H. Gershoni, and Y. T. Radday. Optimization of typewriter keyboard by computer-simulation. *Angewandte Informatik*, 10:438–439, 1976.

R. Poveda and J. Gómez. Solving the quadratic assignment problem (QAP) through a fine-grained parallel genetic algorithm implemented on GPUs. In *International Conference on Computational Collective Intelligence*, pages 145–154. Springer, 2018.

F. P. Preparata and M. I. Shamos. *Computational Geometry: An Introduction*. Springer Science & Business Media, New York, 2012.

A. P. Punnen and Y. Wang. The bipartite quadratic assignment problem and extensions. *European Journal of Operational Research*, 250(3):715–725, 2016.

S. Quader and K. K. Castillo-Villar. Design of an enhanced multi-aisle order-picking system considering storage assignments and routing heuristics. *Robotics and Computer-Integrated Manufacturing*, 50:13–29, 2018.

S. S. Rao and G. K. Adil. Optimal class boundaries, number of aisles, and pick list size for low-level order picking systems. *IIE Transactions*, 45(12):1309–1321, 2013.

S. S. Rao and G. K. Adil. Analytical models for a new turnover-based hybrid storage policy in unit-load warehouses. *International Journal of Production Research*, 55(2):327–346, 2017.

S. S. Rao, G. K. Adil, and R. Venkitasubramony. On the expectation of the largest gap in a warehouse. *Transportation Research Part E: Logistics and Transportation Review*, 143: 102103, 2020.

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

J. Renaud and A. Ruiz. Improving product location and order picking activities in a distribution centre. *Journal of the Operational Research Society*, 59(12):1603–1613, 2008.

J. Reyes, E. Solano-Charris, and J. Montoya-Torres. The storage location assignment problem: A literature review. *International Journal of Industrial Engineering Computations*, 10(2): 199–224, 2019.

A. Rimélé, M. Gamache, M. Gendreau, P. Grangier, and L.-M. Rousseau. Robotic mobile fulfillment systems: A mathematical modelling framework for e-commerce applications. *International Journal of Production Research*, pages 1–17, 2021.

K. J. Roodbergen. Storage assignment for order picking in multiple-block warehouses. In R. Manzini, editor, *Warehousing in the Global Supply Chain: Advanced Models, Tools and Applications for Storage Systems*, pages 139–155. Springer, 2012.

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

K. J. Roodbergen and I. F. A. Vis. A model for warehouse layout. *IIE Transactions*, 38(10): 799–811, 2006.

K. J. Roodbergen and I. F. A. Vis. A survey of literature on automated storage and retrieval systems. *European Journal of Operational Research*, 194(2):343–362, 2009.

K. J. Roodbergen, I. F. A. Vis, and G. D. Taylor Jr. Simultaneous determination of warehouse layout and control policies. *International Journal of Production Research*, 53(11):3306–3326, 2015.

M. B. Rosenwein. An application of cluster analysis to the problem of locating items within a warehouse. *IIE Transactions*, 26(1):101–103, 1994.

B. Rouwenhorst, B. Reuter, V. Stockrahm, G.-J. van Houtum, R. J. Mantel, and W. H. M. Zijm. Warehouse design and control: Framework and literature review. *European Journal of Operational Research*, 122(3):515–533, 2000.

D. Roy, S. Nigam, R. B. M. De Koster, I. Adan, and J. Resing. Robot-storage zone assignment strategies in mobile fulfillment systems. *Transportation Research Part E: Logistics and Transportation Review*, 122:119–142, 2019.

S. Sahni and T. Gonzalez. P-complete approximation problems. *Journal of the Association for Computing Machinery*, 23(3):555–565, 1976.

H. Saito, T. Fujie, T. Matsui, and S. Matuura. A study of the quadratic semi-assignment polytope. *Discrete Optimization*, 6(1):37–50, 2009.

S. Salhi and G. K. Rand. The effect of ignoring routes when locating depots. *European Journal of Operational Research*, 39(2):150–156, 1989.

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

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*, 253(1):68–84, 2016.

A. H. Schrotenboer, S. Wruck, K. J. Roodbergen, M. Veenstra, and A. S. Dijkstra. Order picker routing with product returns and interaction delays. *International Journal of Production Research*, 55(21):6394–6406, 2017.

I. Schüle, H. Ewe, and K.-H. Küfer. Finding tight RLT formulations for quadratic semi-assignment problems. In S. Cafieri, A. Mucherino, G. Nannicini, F. Tarissan, and L. Liberti, editors, *Cologne-Twente Workshop on Graphs and Combinatorial Optimization*, pages 109–112. University of Twente, 2009.

H. D. Sherali and W. P. Adams. Reformulation-linearization techniques for discrete optimization problems. In P. M. Pardalos, D.-Z. Du, and R. L. Graham, editors, *Handbook of Combinatorial Optimization*, pages 479–532. Springer, London, 2013.

Y. Shi, T. Wang, and L. C. Alwan. Analytics for cross-border e-commerce: Inventory risk management of an online fashion retailer. *Decision Sciences*, 2020.

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

J. Skorin-Kapov. Tabu search applied to the quadratic assignment problem. *ORSA Journal on Computing*, 2(1):33–45, 1990.

M. Skutella. Convex quadratic and semidefinite programming relaxations in scheduling. *Journal of the Association for Computing Machinery*, 48(2):206–242, 2001.

J. M. Smith and W.-J. Li. Quadratic assignment problems and M/G/C/C/state dependent network flows. *Journal of Combinatorial Optimization*, 5(4):421–443, 2001.

E. Sonuc, B. Sen, and S. Bayir. A cooperative GPU-based parallel multistart simulated annealing algorithm for quadratic assignment problem. *Engineering Science and Technology, An International Journal*, 21(5):843–849, 2018.

N. Sooksaksun, V. Kachitvichyanukul, and D.-C. Gong. A class-based storage warehouse design using a particle swarm optimisation algorithm. *International Journal of Operational Research*, 13(2):219–237, 2012.

L. Steinberg. The backboard wiring problem: A placement algorithm. *SIAM Review*, 3(1): 37–50, 1961.

T. Stützle. Iterated local search for the quadratic assignment problem. *European Journal of Operational Research*, 174(3):1519–1539, 2006.

É. Taillard. Robust taboo search for the quadratic assignment problem. *Parallel Computing*, 17(4-5):443–455, 1991.

E.-G. Talbi and V. Bachelet. Cosearch: A parallel cooperative metaheuristic. *Journal of Mathematical Modelling and Algorithms*, 5(1):5–22, 2006.

E.-G. Talbi, O. Roux, C. Fonlupt, and D. Robillard. Parallel ant colonies for the quadratic assignment problem. *Future Generation Computer Systems*, 17(4):441–449, 2001.

E. Tappia, D. Roy, M. Melacini, and R. B. M. De Koster. Integrated storage-order picking systems: Technology, performance models, and design insights. *European Journal of Operational Research*, 274(3):947–965, 2019.

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

J. A. Tompkins, J. A. White, Y. A. Bozer, and J. M. A. Tanchoco. *Facilities Planning*. John Wiley & Sons, New York, 4th edition, 2010.

U. Tosun. On the performance of parallel hybrid algorithms for the solution of the quadratic assignment problem. *Engineering Applications of Artificial Intelligence*, 39:267–278, 2015.

S. Tsutsui and N. Fujimoto. Solving quadratic assignment problems by genetic algorithms with GPU computation: A case study. In *Proceedings of the 11th Annual Conference Companion on Genetic and Evolutionary Computation Conference: Late Breaking Papers*, pages 2523–2530, 2009.

C. R. Turner, A. Fuggetta, L. Lavazza, and A. L. Wolf. A conceptual basis for feature engineering. *Journal of Systems and Software*, 49(1):3–15, 1999.

Y. Z. Ünal and Ö. Uysal. A new mixed integer programming model for curriculum balancing: Application to a Turkish university. *European Journal of Operational Research*, 238(1): 339–347, 2014.

C. A. Valle and J. E. Beasley. Order allocation, rack allocation and rack sequencing for pickers in a mobile rack environment. *Computers & Operations Research*, 125:105090, 2021.

J. P. Van den Berg. Class-based storage allocation in a single-command warehouse with space requirement constraints. *International Journal of Industrial Engineering*, 3:21–28, 1996.

T. van Gils, K. Ramaekers, A. Caris, and M. Cools. The use of time series forecasting in zone order picking systems to predict order pickers' workload. *International Journal of Production Research*, 55(21):6380–6393, 2017.

T. Van Gils, A. Caris, and K. Ramaekers. Reducing picker blocking in a high-level narrow-aisle order picking system. In *2018 Winter Simulation Conference (WSC)*, pages 2953–2965. IEEE, 2018a.

T. Van Gils, K. Ramaekers, K. Braekers, B. Depaire, and A. Caris. Increasing order picking efficiency by integrating storage, batching, zone picking, and routing policy decisions. *International Journal of Production Economics*, 197:243–261, 2018b.

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

T. Van Gils, A. Caris, K. Ramaekers, and K. Braekers. Formulating and solving the integrated batching, routing, and picker scheduling problem in a real-life spare parts warehouse. *European Journal of Operational Research*, 277(3):814–830, 2019a.

T. Van Gils, A. Caris, K. Ramaekers, K. Braekers, and R. B. M. de Koster. Designing efficient order picking systems: The effect of real-life features on the relationship among planning problems. *Transportation Research Part E: Logistics and Transportation Review*, 125:47–73, 2019b.

T. Van Luong, N. Melab, and E.-G. Talbi. GPU computing for parallel local search metaheuristic algorithms. *IEEE Transactions on Computers*, 62(1):173–185, 2011.

S. Vanheusden, T. Van Gils, A. Caris, K. Ramaekers, and K. Braekers. Operational workload balancing in manual order picking. *Computers & Industrial Engineering*, 141:106269, 2020.

W. Wang, Y. Wu, J. Zheng, and C. Chi. A comprehensive framework for the design of modular robotic mobile fulfillment systems. *IEEE Access*, 8:13259–13269, 2020.

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

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

F. Weidinger, N. Boysen, and D. Briskorn. Storage assignment with rack-moving mobile robots in KIVA warehouses. *Transportation Science*, 52(6):1479–1495, 2018.

S. Weisberg. *Applied Linear Regression*, volume 528. John Wiley & Sons, Hoboken, NJ, 2013.

M. Wulfraat. Is Kiva systems a good fit for your distribution center? An unbiased distribution consultant evaluation, 2012. URL `https://www.mwpvl.com/html/kiva_systems.html`.

P. R. Wurman, R. D'Andrea, and M. Mountz. Coordinating hundreds of cooperative, autonomous vehicles in warehouses. *AI Magazine*, 29(1):9–9, 2008.

X. Xiang, C. Liu, and L. Miao. Storage assignment and order batching problem in Kiva mobile fulfilment system. *Engineering Optimization*, 50(11):1941–1962, 2018.

J. Xiao and L. Zheng. A correlated storage location assignment problem in a single-block-multi-aisles warehouse considering BOM information. *International Journal of Production Research*, 48(5):1321–1338, 2010.

L. Xie, N. Thieme, R. Krenzler, and H. Li. Introducing split orders and optimizing operational policies in robotic mobile fulfillment systems. *European Journal of Operational Research*, 288(1):80–97, 2021.

T. Xu, P. Yang, and H. Guo. Energy efficiency analysis on robotic mobile fulfillment system. In *2019 IEEE 6th International Conference on Industrial Engineering and Applications (ICIEA)*, pages 145–149. IEEE, 2019.

F. Yener and H. R. Yazgan. Optimal warehouse design: Literature review and case study application. *Computers & Industrial Engineering*, 129:1–13, 2019.

Y. Yu and R. B. M. De Koster. Optimal zone boundaries for two-class-based compact three-dimensional automated storage and retrieval systems. *IIE Transactions*, 41(3):194–208, 2009.

R. Yuan, T. Cezik, and S. C. Graves. Stowage decisions in multi-zone storage systems. *International Journal of Production Research*, 56(1-2):333–343, 2018.

Z. Yuan and Y. Y. Gong. Bot-in-time delivery for robotic mobile fulfillment systems. *IEEE Transactions on Engineering Management*, 64(1):83–93, 2017.

N. Zaerpour, Y. Yu, and R. B. M. De Koster. Optimal two-class-based storage in a live-cube compact storage system. *IISE Transactions*, 49(7):653–668, 2017.

H. Zhang, C. Beltran-Royo, and L. Ma. Solving the quadratic assignment problem by means of general purpose mixed integer linear programming solvers. *Annals of Operations Research*, 207(1):261–278, 2013.

L. Zhou, Y. Shi, J. Wang, and P. Yang. A balanced heuristic mechanism for multirobot task allocation of intelligent warehouses. *Mathematical Problems in Engineering*, 2014, 2014.

W. Zhu, J. Curry, and A. Marquez. SIMD tabu search for the quadratic assignment problem with graphics hardware acceleration. *International Journal of Production Research*, 48(4): 1035–1047, 2010.

Y. Zhuang, Y. Zhou, Y. Yuan, X. Hu, and E. Hassini. Order picking optimization with rack-moving mobile robots and multiple workstations. *European Journal of Operational Research*, 2021.

B. Zou, Y. Gong, X. Xu, and Z. Yuan. Assignment rules in robotic mobile fulfilment systems for online retailers. *International Journal of Production Research*, 55(20):6175–6192, 2017.

B. Zou, X. Xu, Y. Gong, and R. De Koster. Evaluating battery charging and swapping strategies in a robotic mobile fulfillment system. *European Journal of Operational Research*, 267(2):733–753, 2018.