JOINT SOLUTION OF THE ORDER BATCHING, PICKER ROUTING, STORAGE LOCATION ASSIGNMENT, AND SCHEDULING PROBLEMS

by

Ozan Rıdvan Aksu

B.S., Management Engineering, İstanbul Technical University, 2013

M.S., Industrial Engineering, Yıldız Technical University, 2018

Submitted to the Institute for Graduate Studies in
Science and Engineering in partial fulfillment of
the requirements for the degree of
Doctor of Philosophy

Graduate Program in Industrial Engineering

Boğaziçi University

2025

ACKNOWLEDGEMENTS

I would like to express my deepest gratitude to my thesis advisors for their invaluable guidance, encouragement, and unwavering support throughout the course of this research. Their insightful feedback, constructive criticism, and patient mentorship have been instrumental in shaping both this work and my development as a researcher. I am sincerely thankful for the time, expertise, and dedication they generously devoted to my progress, without which this thesis would not have been possible.

ABSTRACT

JOINT SOLUTION OF THE ORDER BATCHING, PICKER ROUTING, STORAGE LOCATION ASSIGNMENT, AND SCHEDULING PROBLEMS

This thesis addresses the optimization of key warehouse order picking operations, focusing on Order Batching Problem, Picker Routing Problem, Location Assignment Problem, and Scheduling Problem. First, we introduce the Joint Order Batching and Routing Policy Selection Problem which involves forming batches and selecting routing policies under a given storage policy. The aim is to minimize total travel distance. We solve this problem optimally using two exact methods; column-and-cut generation and branch-price-and-cut. Second, we extend the analysis to the Joint Order Batching, Batch Sequencing, and Picker Routing Problem with Deadlines, solved using an exact column-and-cut generation method and a heuristic variant of it. Computational experiments on benchmark instances demonstrate that our exact approach outperforms existing methods in both solution quality and computation time for small and medium cases, while the heuristic variant achieves superior results for large instances. The findings provide valuable managerial insights into the benefits of integrating batching, routing, storage assignment, and scheduling decisions in warehouse optimization.

ÖZET

SİPARİŞ GRUPLAMA, TOPLAYICI ROTALAMA, DEPOLAMA YERİ ATAMA VE ÇİZELGELEME PROBLEMLERİNİN BİRLİKTE ÇÖZÜMÜ

Bu tez, depo sipariş toplama operasyonlarının optimizasyonunu ele almakta olup Sipariş Gruplama Problemi, Toplayıcı Rotalama Problemi, Depolama Yeri Atama Problemi ve Çizelgeleme Problemlerine odaklanmaktadır. İlk olarak, belirli bir depolama politikası altında siparişlerin gruplara ayrılması ve rotalama politikalarının seçilmesini içeren Birleşik Sipariş Gruplama ve Rotalama Politikası Seçim Problemi tanıtılmaktadır. Amaç, toplam seyahat mesafesini en aza indirmektir. Bu problem, sütun ve kesme üretimi ile dal-fiyatlandırma-ve-kesme yöntemleri olmak üzere iki kesin yöntem kullanılarak en iyi şekilde çözülmüştür. İkinci olarak, analiz Birleşik Sipariş Gruplama, Parti Sıralama ve Teslim Tarihli Toplayıcı Rotalama Problemine genişletilmiş; bu problem, kesin bir sütun ve kesme üretimi yöntemi ile onun sezgisel bir varyantı kullanılarak çözülmüştür. Literatürde yer alan test örnekleri üzerinde yapılan hesaplama deneyleri, kesin yaklaşımımızın küçük ve orta ölçekli durumlarda hem çözüm kalitesi hem de hesaplama süresi açısından mevcut yöntemlerden daha iyi sonuçlar verdiğini, sezgisel yöntemin ise büyük ölçekli durumlarda üstün performans sergilediğini göstermektedir. Bulgular, depo optimizasyonunda sipariş gruplama, rotalama, depolama yeri atama ve çizelgeleme kararlarının bütünleşik olarak ele alınmasının faydalarına dair önemli yönetsel içgörüler sunmaktadır.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF SYMBOLS

A	Set of arcs
a_{ob}	Equal to one if order o is in batch b
$\bar{a}_o$	Equal to one if batch of pricing sub-problem contains order o
$\mathcal{B}$	Set of batches
C	Batch capacity
$\tilde{c}$	Reduced cost of a label
c_o	Size of order o
$\mathcal{D}$	Set of deadlines
d_k	Completion time of route k
$\bar{d}_o$	Deadline of order o
d_r^b	Distance of batch b with respect to routing policy r
e	Backward label
f	Fractional part of the number of batches
G	Graph
$\mathcal{G}$	Set of SRIs
INS_{MO}	First set of instances
INS_{WG}	Second set of instances
$\mathcal{K}$	Set of routes
K	Represents the number of batches
L	Picking aisle length
ℓ	Label
$\mathcal{O}$	Set of orders
$\mathcal{P}$	Set of pickers
$\mathcal{Q}$	Set of integers
$\mathcal{R}$	Set of routing policies
$\mathcal{S}$	Subset of orders
$\mathcal{T}$	Set of CCs
u	Storage location length

V	Set of nodes
w	Distance between two consecutive picking aisles
x_k^p	Equal to one if picker p performs route k
x_r^b	Equal to one if batch b and routing policy r are selected
$\bar{x}_r^b$	Optimal solution to the RMP
Y	Cross aisle width
y_o	Equal to one if order o is in the searched subset
γ	Minimum number of batches
ϵ	Set of undominated backward labels
θ	Size of the label pool
λ_r^b	Equal to one if batch b of routing policy r has any order
μ	Dual price of the order constraints
$\xi_{\bar{d}}^p$	Dual price of deadline constraints
π	Total dual price of a label
ρ	Dual price of CC
σ	Dual price of SRI
υ	Dual price of MTC

LIST OF ACRONYMS/ABBREVIATIONS

2D	Two Dimensional
3D	Three Dimensional
ALNS	Adaptive Large Neighbourhood Search
BC	Branch-and-Cut
BP	Branch-and-Price
BPC	Branch-Price-and-Cut
CC	Capacity Cut
CC1	Capacity Cuts Using First Separation
CC2	Capacity Cuts Using Second Separation
CC3	Capacity Cuts Using Third Separation
CCG	Column-and-Cut Generation
CCG-H	Heuristic variant of Column-and-Cut Generation
CCVRP	Cumulative Capacitated Vehicle Routing Problem
CG	Column Generation
CGH	Column Generation Heuristic
CPU	Central Processing Unit
CVRP	Capacitated Vehicle Routing Problem
DHOBP	Dedicated Heterogeneous Order Batching Problem
ILS	Iterated Local Search
JOBBSPRP-D	Joint Order Batching, Batch Sequencing and Picker Routing Problem with Deadlines
JOBPRP	Joint Order Batching and Picker Routing Problem
JOBRPSP	Joint Order Batching and Routing Policy Selection Problem
LAP	Location Assignment Problem
LB	Lower Bound
MDVRP	Multi-Depot Vehicle Routing Problem
MILP	Mixed Integer Linear Programming
MIP	Mixed Integer Programming

MÖ	Muter and Öncan
MP	Master Problem
MTC	Martello and Toth's Cut
NEH	Nawaz, Enscore, and Ham Heuristic
OBP	Order Batching Problem
PRP	Picker Routing Problem
PSA-ACO	Parallel Simulated Annealing and Ant Colony Optimization
PSP	Pricing Sub-Problem
REEVRP	Range-Extended Electric Vehicle Routing Problem
RMP	Restricted Master Problem
SP	Scheduling Problem
SRI	Subset Row Inequality
TSP	Traveling Salesman Problem
UB	Upper Bound
VNS	Variable Neighborhood Search
VRPB	Vehicle Routing Problems with Backhouls
VRPTW	Vehicle Routing Problem with Time Windows
WG	Wahlen and Gschwind

1. INTRODUCTION

According to Tompkins et al. [1] and Bartholdi and Hackman [2], order picking accounts for approximately 55% of a warehouse's total operating cost, while Coyle et al. [3] estimate this figure to be between 60% and 70%. Order picking involves several interrelated operations on different levels of planning horizon, and in this study, we focus on a subset of these. The Order Batching Problem (OBP), Picker Routing Problem (PRP), and Scheduling Problem (SP) require decisions to be made strategically, while Location Assignment Problem (LAP) belongs to the group of tactical decisions according to Xie et al. [4].

An order consists of a list of items that pickers need to collect. Rather than picking each order individually, pickers can combine multiple orders—a process known as batching—where the combined set of orders assigned to a picker is referred to as a batch. The number of items in any batch must not exceed the picker's capacity. The task of grouping orders into batches to minimize the total picking time is defined as the OBP.

Once the pickers have the list of items to be collected, their routes need to be decided. One option is to find an optimal tour for each picker, which will yield the minimum picking time or the cost thereof. There are also several routing policies, in which the pickers follow certain traveling rules. In the literature, there are six picker routing policies: traversal, return, midpoint, largest gap, composite and mixed [5]. These policies may result in near optimal picking times. Finding the best tour for the pickers is referred to as the PRP.

The third problem is the LAP. The assignment of the items has a big impact on the length of the tours. Finding the optimal locations of each item while solving the OBP requires a large amout of computational time. There are also some storage polices which follow certain rules when locating the items. The most commonly studied ones

are random, closest open location, dedicated, full turnover, and class based storage policies [6].

SP deals with the completion times of the orders. They involve makespan minimization, earliness and tardiness minimization, batching with deadlines. Makespan is the completion time of all batches. Certain orders might have deadlines. In order to complete the collection of these orders on time, earliness or tardiness of completion times of these orders can be minimized. When the deadlines are strongly restrictive, they can be added to the model as hard constraints.

The first aim of this study is to solve the Joint Order Batching and Routing Policy Selection Problem (JOBRPSP) with a given storage policy. The JOBPRSP problem involves forming order batches and selecting a route for each batch from a set of alternatives, with the goal of minimizing the total distance traveled by pickers, under a given storage policy. To the best of our knowledge, this specific problem extension has not been previously studied in the literature. We focus on the three most commonly used routing heuristics: return, traversal, and largest gap. To solve the problem optimally, we propose two exact solution methods: one is based on column-and-cut generation, and the other one is using a branch-price-and-cut approach. These methods incorporate two different labeling algorithms and employ two valid inequalities previously introduced for the OBP. We assess the impact of using multiple routing policies versus a single routing policy or optimal routing. Additionally, we examine the influence of five storage assignment strategies: random, across-aisle, within-aisle, perimeter, and diagonal. A variety of problem instances, built on existing benchmarks, are solved to optimality. Finally, we provide managerial insights derived from the computational results.

Secondly, OBP will be solved by considering the scheduling aspect of warehouse optimization. The problem is referred to as Joint Order Batching, Batch Sequencing and Picker Routing Problem with Deadlines (JOBBSPRP-D), and solved by CCG method. Alongside the exact CCG method, which is effective for solving small to

medium-sized instances of the JOBBSPRP-D, we also introduce a heuristic variant called CCG-H to handle larger instances. Both approaches are evaluated using benchmark instances from the literature and compared against two existing methods: a column generation heuristic (CGH) and an iterated local search (ILS) metaheuristic. Computational experiments demonstrate that the exact CCG method outperforms CGH in 26 instances with 100 orders and a batch capacity of 15, delivering better feasible solutions in 12 of those cases (11 of which are optimal) and achieving optimality in less time for 13 instances. Furthermore, the heuristic CCG-H method yields better feasible solutions in 16 out of 27 instances with 100 orders and a batch capacity of 30, outperforming the existing algorithms.

In the next section, we review some of the studies on warehouse optimization. In Chapter 3 we introduce the warehouse layout, model assumptions, routing policies and storage policies. Chapter 4 involves the decomposition approach to the same problem and further review on the literature about the solution methods. In Chapter 5 we discuss on the results of the computational experiments. The method developed to solve JOBBSPRP-D is introduced in Chapter 6 and the results are presented in Chapter 7. Finally, the conclusion of this thesis will be presented in Chapter 8.

2. LITERATURE REVIEW

In this section we review the warehouse optimization studies in two groups. The first group of studies jointly solves order batching with another type of problem such as picker routing or location assignment/storage policy selection. The second group consists of scheduling problems.

2.1. Order Batching, Picker Routing, Location Assignment Problems

In the literature, there are several heuristic algorithms used for OBP. Scholz and Wascher [7] combine an iterative local search heuristic for OBP with routing algorithms. Kulak et al. [8] propose cluster based tabu search algorithms for Joint Order Batching and Picker Routing Problem (JOBPRP). Their heuristics first create batches and then solve a Traveling Salesman Problem (TSP) for these batches. The authors are able to solve instances with 250 orders in about 30 seconds. Öncan [5] proposes Mixed Integer Linear Programming (MILP) formulations of OBP for traversal, return and midpoint routing policies, and develop an ILS algorithm. In a more recent study, Oxenstierna et al. [9] compare two heuristic algorithms on OBP where the weight and volume of the products are considered. Wagner and Mönch [10] introduce the dedicated heterogeneous OBP (DHOBP) in which they consider a finite number of pick devices with different capacities, propose two Variable Neighborhood Search (VNS) schemes, and for a given routing policy test them on newly generated and benchmark instances.

Gademann and Velde [11] develop a Branch-and-Price (BP) algorithm for the OBP. Valle et al. [12] propose a branch-and-cut (BC) algorithm for the JOBPRP and compare this algorithm with solving a multi-commodity network flow formulation. The latter performs better, yet fails to solve instances with more than 13 orders. Valle et al. [13] devise another BC algorithm and generate additional valid cuts to the JOBPRP. Briant et al. [14] develop a Column Generation Heuristic (CGH) algorithm for the same problem and compare their results with those in [13]. The authors show that

their CGH performs better on instances with orders up to 30. Jamal et al. [15] suggest a mathematical model for JOBPRP where they consider the skill of each picker on collecting items of various weight and volume, from locations of various height. They minimize the total distance travelled and the effort of the pickers. Heßlera and Irnicha [16] present a Branch-Price-and-Cut (BPC) algorithm for the JOBPRP with scatter storage in which the products are stored at more than one location in general. Chabot et al. [17] develop an Adaptive Large Neighbourhood Search (ALNS) heuristic and a BC for PRP in a warehouse with two-dimensional aisles in which items can be stored on different height levels of an aisle.

Bahçeci and Öncan [18] devise MILP formulations for composite, largest gap, optimal and mixed routing policies. They also compare the performance of these routing policies over five different storage policies. Yang [19] focus on the effects of storage assignment and order batching policies in a parts-to-pickers system. Kübler et al. [20] develop a heuristic method for JOBPRP and also calculate the reduction of the total travelled distance that may result from relocating the items, based on the forecast of next periods' demand values. Yang et al. [21] analyze the effect of storage assignment policies with a model which aims to determine the optimal layout of a warehouse.

There are several studies in the literature that use simulation techniques to analyze the interaction between storage and routing policies. Caron et al. [22], Petersen and Schmenner [23], Petersen and Aase [24], Dekker et al. [25], Hsieh and Tsai [26], Dukic and Oluic [27], Chen et al. [28], Chan and Chan [29], Hsieh and Huang [30], Shqair et al. [31], Franzke et al. [32], and Van Gils et al. [33] exclude the decision of order batching and point out the best combinations of routing and storage policies.

A summary of the literature review on order batching, picker routing and location assignment problems is presented with Table 2.1.

Table 2.1. Literature review on order batching, picker routing and location assignment problems.

Author(s)	Problem Considered	Routing Policy	Solution Approach	Layout
Scholz and Wascher [7]	Order Batching	Traversal, Largest gap, Aisle-by-aisle, Combined	Iterative Local Search	Multi-block
Kulak et al. [8]	Joint Order Batching and Picker Routing	Optimal	Cluster-Based Tabu Search	Single-block
Öncan [5]	Order Batching	Traversal, Return, Midpoint	MILP, Iterated Local Search	Single-block
Oxenstierna et al. [9]	Order Batching	Optimal	Seed Heuristics, Metropolis Algorithm	Single-block
Wagner and Mönch [10]	Order Batching	S-shape, Largest gap	Variable Neighborhood Search	Single-block
Gademann and Velde [11]	Order Batching	Optimal	Branch-and-Price	Single-block
Valle et al. [12]	Joint Order Batching and Picker Routing	Optimal	Branch-and-Cut	Multi-block
Briant et al. [14]	Joint Order Batching and Picker Routing	Optimal	Column Generation	Multi-block
Jamal et al. [15]	Joint Order Batching and Picker Routing	Optimal	MILP	Single-block
Heßler and Irnich [16]	Joint Order Batching and Picker Routing	Optimal	Branch-Price-and-Cut	Single-block
Chabot et al. [17]	Capacitated Narrow Aisle Order Picking	Optimal	Branch-and-Cut, Adaptive Large Neighbourhood Search	Single-block
Haouassi et al. [34]	Joint Order Splitting, Picker Routing, Batch Scheduling	Optimal	MILP, Route First-Schedule Second Heuristic	Multi-block
Bahçeci and Öncan [18]	Order Batching and Storage Policy Selection	Composite, Largest gap, Optimal, Mixed	MILP	Single-block
Yang [21]	Order Batching and Storage Policy Selection	No Routing (Parts to pickers)	MILP	Single-block (3D)
Kübler et al. [20]	Joint Order Batching, Picker Routing, Location Assignment	S-shape	Discrete Evolutionary Particle Swarm Optimization	Multi-block
Yang et al. [19]	Storage Policy Selection	No Routing (Pick-and-pass)	MILP	Multi-block
Muter and Öncan [35]	Order Batching	Traversal, Return, Midpoint	Column Generation	Single-block
Muter and Öncan [36]	Order Batching and Picker Scheduling	Traversal, Return, Midpoint	Column Generation	Single-block
Wahlen and Gschwind [37]	Order Batching	Traversal, Return, Midpoint, Largest gap, Combined, Optimal	Branch-Price-and-Cut	Single-block

2.2. Scheduling Problems

Elsayed and Lee [38] present a mathematical formulation for the order batching problem with the objective of minimizing tardiness and complement their work with simulation-based experiments. The completion times of order picking in batches are calculated by solving a Traveling Salesman Problem for each batch. Chen et al. [39] address the same problem by integrating heuristic methods, where order batching and sequencing decisions are handled using a genetic algorithm, and routing for each batch is determined via ant colony optimization. Zulj et al. [40] propose a hybrid heuristic method combining adaptive large neighborhood search and the Nawaz, Enscore, and Ham heuristic (ALNS/NEH) to minimize tardiness. In their framework, pickers drop off collected items at handover points located in each aisle, from where autonomous mobile robots retrieve them. As such, routing is excluded from their study.

Henn [41] focus on online order batching with the objective of minimizing makespan using a single picker. The author adapts heuristics originally developed for offline order batching for the online context. Ardjmand et al. [42] devise a MILP model and propose a hybrid heuristic combining parallel simulated annealing and ant colony optimization (PSA-ACO) for the order batching and makespan minimization problem in a wave picking system with multiple pickers. In a follow-up study, Ardjmand et al. [43] introduce a column generation (CG) heuristic for the same problem and compare its efficiency with that of PSA-ACO. Their CG approach solves the pricing subproblem using a genetic algorithm, while an artificial neural network estimates the optimal travel times for batches. Muter and Öncan [36] aim to minimize makespan along with total travel distance using an exact CG-based algorithm. They treat makespan as a strict deadline, applying it as a hard constraint on the travel time per picker, and iteratively reducing the deadline until it either reaches a lower bound or infeasibility occurs. Saylam et al. [44] utilize dynamic programming to minimize makespan in a multi-block warehouse. For each picking wave, they dynamically determine picking zones, defined as unions of adjacent aisles, and incorporate routing decisions.

Cano et al. [45] formulate four mathematical models addressing different versions of the integrated order batching, sequencing, and routing problem. These models target tardiness minimization, include time window constraints, and support multiple pickers with heterogeneous vehicles in both 2D and 3D multi-block warehouses. They highlight the exponential growth of decision variables as the number of item locations increases. Rijal et al. [46] propose a BP approach for the OBP with time windows. Given the high complexity of the problem, the BP algorithm struggles to find optimal solutions for over half of the test instances. Consequently, the authors also design a metaheuristic that generates initial solutions using the savings algorithm and enhances them through large neighborhood search with simulated annealing. Vanheusden et al. [47] investigate how balancing the workload between picking zones can improve efficiency, omitting batching and routing from their model. They evaluate four objective functions in their mathematical model and compare them with an ILS algorithm.

The works most closely related to our study are Van Gils et al. [48] and Briant et al. [49]. The former introduces an ILS algorithm for the JOBBSPRP-D, while the latter proposes a CGH for the same problem. Their evaluation is conducted using the benchmark instances from Van Gils et al. [48]. The authors demonstrate that the CGH heuristic yields superior upper bounds compared to the ILS. However, their method relies on solving the pricing problem with a mixed integer programming (MIP) solver, which proves inefficient; they report that this step consumes 95% of the total computation time. Haouassi et al. [34] analyze the benefits of order splitting for an Order Batching, Batch Scheduling and Picker Routing Problem on the same instances with Van Gils et al. [48]. Order splitting means that an order can be assigned partially to multiple pickers. The authors report that order picking times are reduced by 30% on average.

A summary of the literature review on scheduling problems is presented with Table 2.2.

Table 2.2. Literature review on scheduling problems.

Author(s)	Problem Considered	Routing Policy	Solution Algorithm	Layout
Elyased and Lee [38]	Tardiness	No routing	Simulation	Single-block
Chen et al. [39]	Tardiness	No routing	Genetic Algorithm	Multi-block
Zulj et al. [40]	Tardiness	No routing	Adaptive Large Neighborhood Search	Single-block
Henn [41]	Makespan	Traversal, Largest gap	Heuristic	Single-block
Ardjmand et al. [42]	Makespan	Optimal	Ant Colony Optimization	Single-block
Ardjmand et al. [43]	Makespan	Optimal	Column Generation	Single-block
Muter and Öncan [36]	Makespan	Traversal, Return, Midpoint	Column Generation	Single-block
Saylam et al. [44]	Makespan	Optimal	Dynamic Programming	Multi-block
Cano et al. [45]	Time windows	Optimal	MILP	Multi-block
Rijal et al. [46]	Time windows	Optimal	LNS with Simulated Annealing	Single-block
Van Gils et al. [48]	Deadlines	Optimal	Iterated Local Search	Multi-block
Briant et al. [49]	Deadlines	Optimal	Column Generation	Multi-block
Haouassi et al. [34]	Deadlines	Optimal	Route First-Schedule Second Heuristic	Multi-block

3. BACKGROUND INFORMATION ON WAREHOUSE OPTIMIZATION

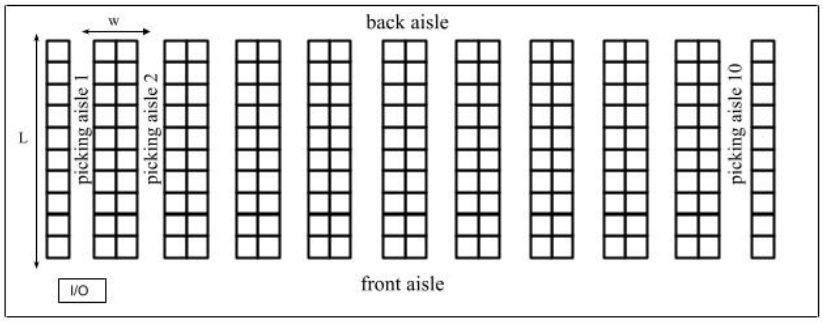
In this section, we describe the single-block and two-block warehouse layouts, model assumptions, routing policies, and storage policies.

3.1. Warehouse Layout

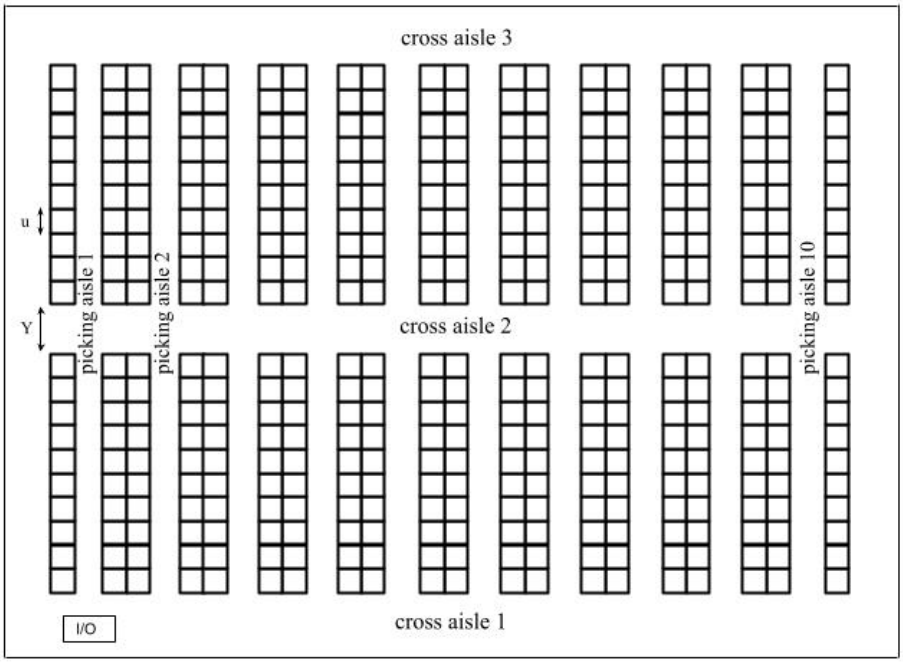
As illustrated with Figure 3.1, we focus on rectangular single-block and two-block designs featuring parallel picking aisles, cross aisles, and a depot (labeled "I/O") located in front of the leftmost picking aisle. Figure 3.1(a) is a single block warehouse layout and Figure 3.1(b) is a two block warehouse layout. The picking aisles are wide enough for a picker to access both sides while moving along an aisle. In the figure, "L" denotes the length of an aisle—the distance between the front and back cross aisles—while "w" indicates the spacing between the center lines of two adjacent picking aisles. "Y" is the cross-aisle width and "u" is the pick location length. Each picker starts the tour from the depot, collects the items in their batch and returns to the depot at the end of the tour. The number of items in a batch cannot exceed a capacity limit.

3.2. Model Assumptions

Order splitting is not allowed in our model. In other words, all items of an order must be picked by the same picker. Each item is located in one storage location and its capacity is large enough to locate as many items as desired. The aisles are narrow so that when in a picking aisle, a picker can reach both sides without difficulty.



(a) Single-block warehouse.



(b) Two-block warehouse.

Figure 3.1. Layout of rectangular single-block and two-block warehouses.

3.3. Routing Policies for Single-Block Warehouse

At this stage we will introduce well-known heuristic routing policies in the literature which are return, traversal, midpoint, largest gap, composite and mixed.

Return: In the return routing policy, the picker moves along the front aisle until the location of the item that needs to be picked is reached. The picker reaches the

picking aisles using only the front aisle. After picking the last item at the right-most aisle, the picker returns to the depot.

Traversal: In the traversal routing policy, the picker traverses the entire picking aisle in which an item to be collected is located. The picker enters the aisle from front aisle and exits through the back aisle, or vice versa. If the number of aisles which should be visited is odd, the picker enters the rightmost aisle from the front aisle, moves until the last item to be picked, and then makes a U-turn and reaches to the front aisle from where they go back to the depot.

Midpoint: In the midpoint routing policy, the picker traverses the first aisle to be visited and reaches the back aisle. For the remaining aisles in which items to be picked are located, the picker only collects the items which are closer to the back aisle than front aisle. When on the rightmost picking aisle where at least one item is retrieved, the picker traverses again and exits from the front aisle to collect the remaining items which are located closer to the front aisle than the back aisle. At the end, the picker reaches the depot and finishes the route.

Largest Gap: Likewise the midpoint routing policy, in the largest gap routing policy, the leftmost and rightmost picking aisles are traversed entirely and the picker exits from the back, and front aisles, respectively. For the remaining aisles, the largest gap rule is applied. The largest gap in an aisle can be between two adjacent items, or an item and front or back aisle. When the largest gap is between two adjacent items, the picker enters this aisle from both back and front aisles to collect the items. Otherwise, the picker enters this aisle from where the largest gap is at the opposite aisle (back or front).

Composite: In the composite routing policy, after collecting the items in the first aisle, the distances from the current position to the items in the next aisle which are closest to the front and back aisles are calculated. The shorter distance is selected to arrive the next aisle, and the remaining items in this aisle are collected. After collecting

the items in the rightmost aisle, the picker travels to the front aisle in order to return to the depot.

Mixed: In the mixed routing policy, after traversing the first aisle, picker enters the following aisles from front or back aisle by considering the closest items to the front and back aisles.

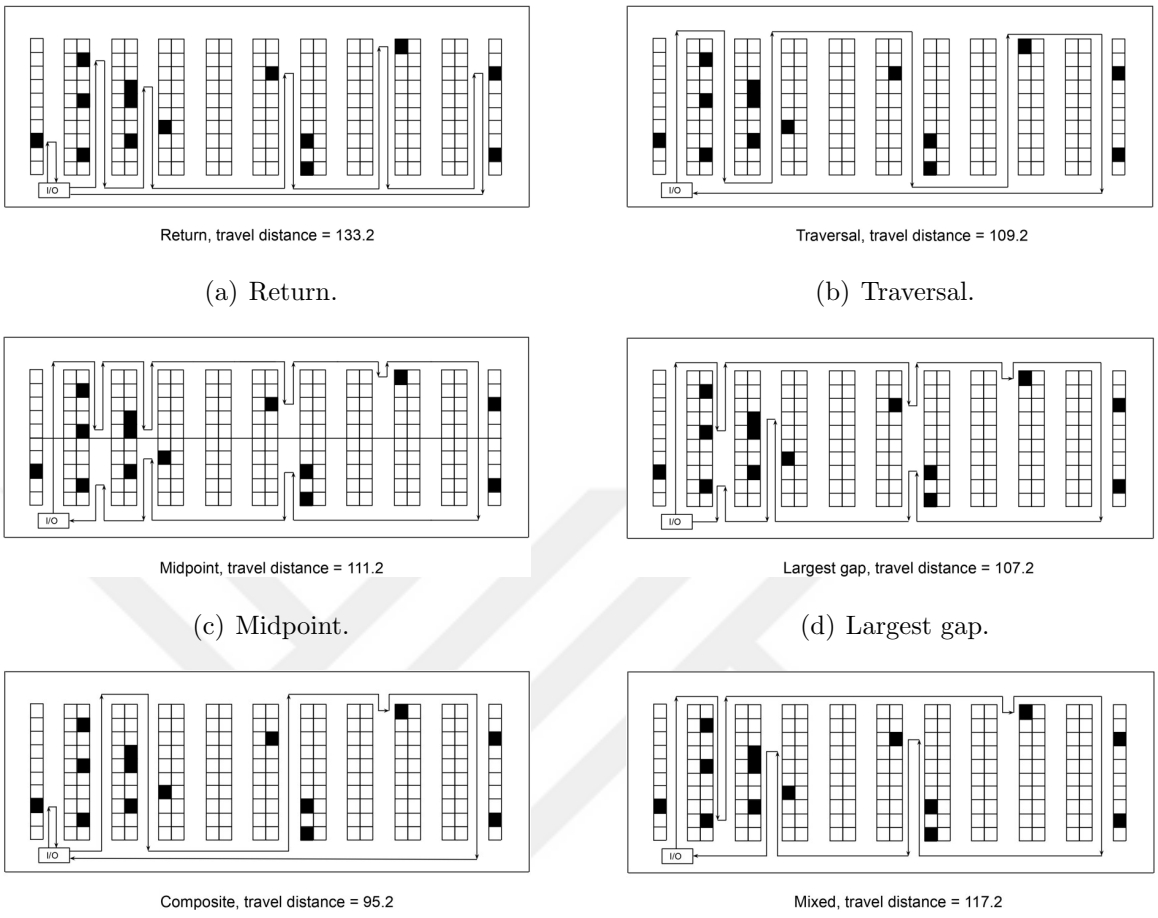
The above-mentioned routing policies for a single-block warehouse are depicted with Figure 3.2. Routing policies return, traversal, midpoint, largest gap, composite, and mixed are illustrated with Figure 3.2(a), Figure 3.2(b), Figure 3.2(c), Figure 3.2(d), Figure 3.2(e), and Figure 3.2(f), respectively.

As stated in Korbacher et al. [50], the largest gap routing policy dominates mixed and mid-point routing policies, meaning, the former generates shorter routes for any batch. In addition to that, composite routing is not a monotone routing policy. As shown by Wahlen and Gschwind [37], when a new item is added to a batch, it may result in a reduction in the distance of the batch. This creates a problem with the labeling algorithm that is used in column generation procedure, which is explained in the following chapter. Therefore, these routing policies are excluded from our study and we consider the return, traversal and largest gap routing policies.

In order to evaluate the performance of the aforementioned routing policies, in our experiments we also use the optimal routing policy. Ratliff and Rosenthall [51] develop a dynamic programming algorithm to find the optimal route in reasonable computation times for single-block warehouses.

3.4. Routing Policies for Two-Block Warehouse

For a two-block warehouse, the steps of the routing policies are slightly modified. The descriptions are given below assuming that the second block must be visited.



(a) Return. (b) Traversal.

(c) Midpoint. (d) Largest gap.

(e) Composite. (f) Mixed.

Figure 3.2. Comparison of six routing heuristics: Return, Traversal, Midpoint, Largest Gap, Composite, and Mixed.

Return: The left-most picking aisle that must be visited (either in the first or second block) is determined. The picker moves to the second cross aisle, and the picking aisles of the second block is visited with respect to the return routing policy rule. After completing the second block, the right-most picking aisle in the first block that must be visited is determined. The picker traverses entire picking aisle and reaches the first cross aisle. From there, the remaining picking aisles in the first block are visited with respect to the return routing policy rule. Finally, the picker reaches the depot.

Traversal: The left-most picking aisle that must be visited (either in the first or second block) is determined. The picker moves to the second cross aisle. Picking aisles

that must be visited are traversed entirely. If the number of picking aisles that must be visited is odd, the right-most picking aisle is visited according to the return routing policy. When the second block is completed and the picker reaches the second cross aisle, the right-most picking aisle that must be visited in the first block is traversed and the picker reaches the first cross aisle. From there on, the remaining picking aisles that involves an item to be picked are traversed as the traversal routing policy dictates. If the number of picking aisles to be visited is odd, the left-most picking aisle is visited according to the return routing policy. Finally, the depot is reached.

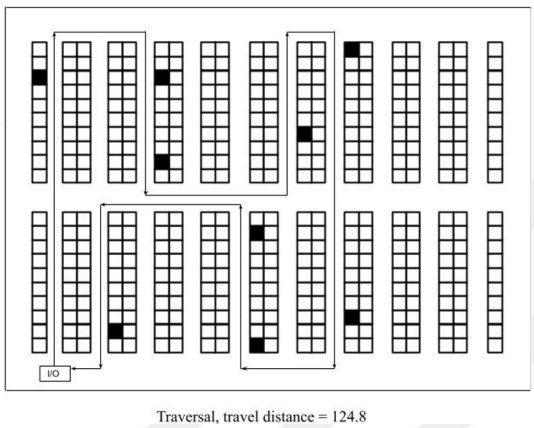
Largest Gap: The left-most picking aisle that must be visited (either in the first or second block) is determined. The picker moves to the second cross aisle. The left-most picking aisle is traversed entirely and the third cross aisle is reached. The remaining picking aisles that must be visited (except the right-most picking aisle) are visited according to the largest gap routing policy rule. The right-most picking aisle is traversed entirely and the second cross aisle is reached. From there on, the left-most picking aisle on the first block that must be entered from the second cross aisle according to the largest gap routing policy rule is reached. The picking aisles of the first block that involves an item is visited with respect to the largest gap routing policy, except the right-most picking aisle. The right-most picking aisle is traversed entirely to reach the first cross aisle. Finally the depot is reached.

The routing policies for a two-block warehouse is represented in Figure 3.3. Routing policies traversal, largest gap, and return are illustrated with Figure 3.3(a), Figure 3.3(b), and Figure 3.3(c), respectively.

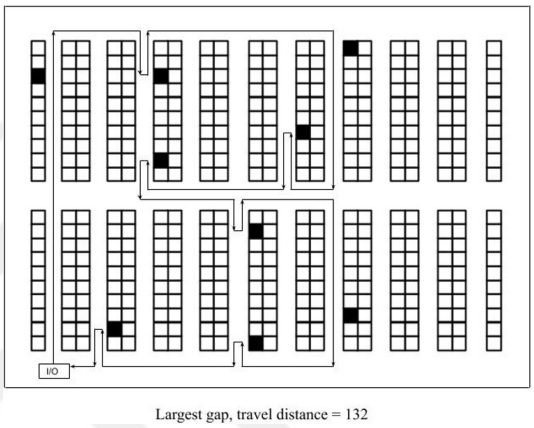
3.5. Storage Policies

In this thesis, we address random storage policy and four well-known class-based storage policies which are across-aisle, within-aisle, perimeter and diagonal storage policies. Class-based storage policies locate the items to the locations with respect to their demand volume. Items are categorized as Class A, Class B and Class C by

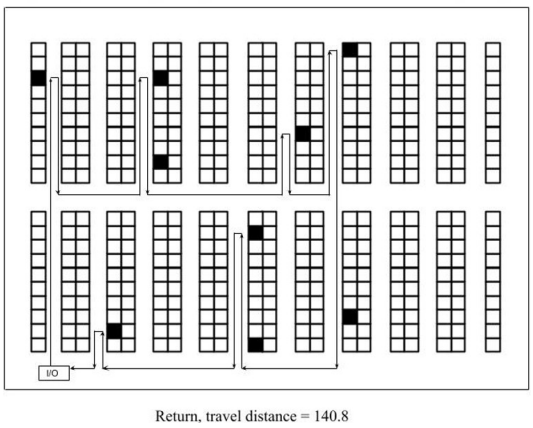
considering the Pareto rule which states that 20% (80%) of the items constitute 80% (20%) of the demand and these items belong to Class A (C). The remaining items are in Class B. Class-based storage policies are depicted with Figure 3.4. Figure 3.4(a), Figure 3.4(b), Figure 3.4(c), and Figure 3.4(d) represent the storage policies within-aisle, diagonal, across-aisle, and perimeter, respectively.



(a) Traversal.

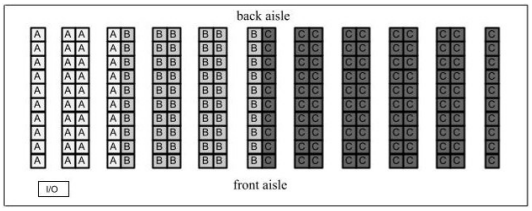


(b) Largest gap.

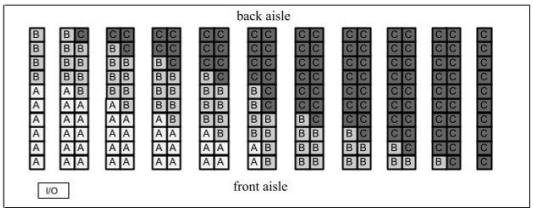


(c) Return.

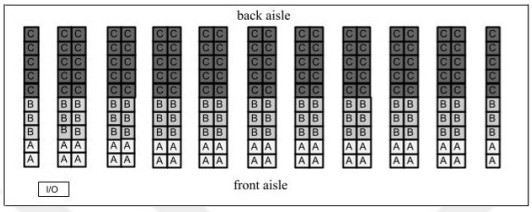
Figure 3.3. Comparison of three routing heuristics: Traversal 2, Largest Gap 2, and Return 2.



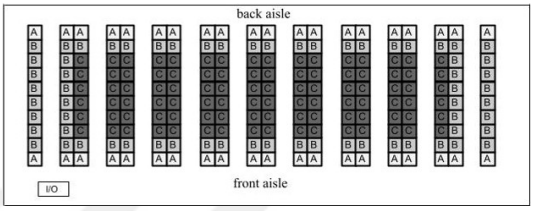
(a) Within-aisle.



(b) Diagonal.



(c) Across-aisle.



(d) Perimeter.

Figure 3.4. Class-based storage policies.

4. COLUMN GENERATION APPROACH TO SOLVE JOINT ORDER BATCHING AND ROUTING POLICY SELECTION PROBLEM

In order to solve the JOBRPSP in a reasonable amount of time, a column generation (CG) algorithm is developed. Given a storage policy, the objective is to minimize the total distance traveled by the pickers to collect the formed batches while selecting the best routing policy for each batch among several alternatives, namely return, traversal and largest gap. Note that CG is used when the number of decision variables is very large and most of these variables have the value of zero in the optimal solution. The problem is then decomposed into two problems: Master Problem (MP) and the Pricing Sub-Problem (PSP).

4.1. Master Problem

We start by defining the following two parameters. The first one, d_r^b, indicates the total distance traveled by a picker to collect all items included in the batch $b \in \mathcal{B}$ using the routing policy $r \in \mathcal{R}$ for a given storage policy. The second parameter a_{ob} equals one if and only if the order $o \in \mathcal{O}$ is included within the batch b. Let the binary decision variable x_r^b be equal to one if and only if the batch b is selected and its items are collected using the routing policy r. Then, the MP, which is a Set Covering Problem (SCP) type formulation of the JOBRPSP is given as

$$\min \sum_{b \in \mathcal{B}} \sum_{r \in \mathcal{R}} d_r^b x_r^b, \tag{4.1}$$

subject to

$$\sum_{r \in \mathcal{R}} \sum_{b \in \mathcal{B}} a_{ob} x_r^b \geq 1 \qquad o \in \mathcal{O}, \tag{4.2}$$

$$x_r^b \in \{0,1\} \qquad r \in \mathcal{R},\ b \in \mathcal{B}, \tag{4.3}$$

where the objective function (4.1) is to minimize the total distance traveled by the pickers. The constraint set (4.2) ensures that each order is included within at least one

batch. Note that the optimal solution of the JOBRPSP provides not only the optimal batches, but also the best routing policy for each batch. Since the generation of all batches requires excessive computational time, we have resorted to employ a CG based solution scheme. To this end, we start with a few columns and solve the LP relaxation of the JOBRPSP model (4.1)–(4.3), which is the so-called restricted master problem (RMP).

4.2. Initial Columns

The selection of the initial columns of RMP can affect the performance of the algorithm. A high quality initial solution can lead to faster convergence to the optimal value of RMP. For this purpose, a greedy savings algorithm is implemented. We basically start with a new batch with a single order. Let the total distance of this batch be $d_{current}$. d_{new} is the distance when a new order is included in the batch and $d_{separate}$ is the distance of the order when it is picked separately. The savings of inclusion of this order is $d_{separate} - (d_{new} - d_{current})$. The unassigned order with maximum savings value is included in the batch if it is feasible with respect to the batch capacity. These steps are repeated until no order is left unassigned.

4.3. Pricing Sub-Problem

During the execution of the CG algorithm, new columns with negative reduced cost are added to the RMP. These columns are generated by solving the PSP considering routing policy r (PSP_r). Let μ_o stand for the dual variable values corresponding to the constraint set (4.2) of the LP relaxation of (4.1)–(4.3) and let the binary decision variable $\overline{a}_o$ equal to one if and only if the batch contains order o. Furthermore, let $\mathcal{O}^+$ be the set of orders included within the batch, i.e. $\mathcal{O}^+ = \{o \in \mathcal{O} : \overline{a}_o = 1\}$. Then, the reduced cost of the batch that incorporates the orders in the set $\mathcal{O}^+$ is indicated by $d_r(O^+) - \sum_{o \in \mathcal{O}} \mu_o \overline{a}_o$, where $d_r(\mathcal{O}^+)$ is a function that calculates the total distance required to retrieve all orders $\mathcal{O}^+$ in the batch using the routing policy r. Let the parameters c_o and C stand for the number of items in the order o and the batch capacity

(picker capacity), respectively. Therefore, the PSP_r for a given routing policy is given as

$$\min d_r(\mathcal{O}^+) - \sum_{o \in \mathcal{O}} \mu_o \overline{a}_o, \tag{4.4}$$

subject to

$$\sum_{o \in \mathcal{O}} c_o \overline{a}_o \leq C, \tag{4.5}$$

$$\overline{a}_o \in \{0, 1\} \qquad o \in \mathcal{O}. \tag{4.6}$$

Note that the pricing subproblems associated with different routing policies should be solved separately. That is, PSP_r has to be solved for every $r \in \mathcal{R}$. Bahçeci and Öncan [18], and Öncan [5] devised mathematical programming models for OBP considering various routing policies. However, since solving the MILP formulation of the PSP may require an excessive amount of CPU time, we opt to implement a labeling algorithm for efficiency. We consider the labeling algorithms developed by Muter and Öncan [35], and Wahlen and Gschwind [37]. The CG algorithm is illustrated with Figure 4.1.

4.4. Labeling Algorithms

Solving the PSP at each iteration significantly deteriorates the efficiency of the CG algorithm. Therefore, Muter and Öncan [35] propose an exact enumeration algorithm, which is called the MÖ labeling algorithm in the sequel, to solve their PSP that occurs as a subproblem within the OBP. The PSP is represented on an acyclic directed graph $G = (V, A)$ where V stands for the node set and A denotes the arc set. The node set V consists of an artificial node and one node for each order. These nodes are sorted in such a way that $\mu_o/c_o \geq \mu_{o+1}/c_{o+1}$ holds, i.e., orders are sorted in non-increasing order with respect to the ratio of the dual variable value to the order volume. This enables the algorithm to produce columns with negative reduced cost in earlier iterations of the CG algorithm. The arc set is defined as $A = \{(i, j) : i, j \in V, i < j\}$. A label is created at the artificial node, and it is then extended to the neighboring nodes through arcs at each iteration if certain criteria are met. The label of a node represents a batch associated with the path p that starts at the artificial node and ends at that node. It

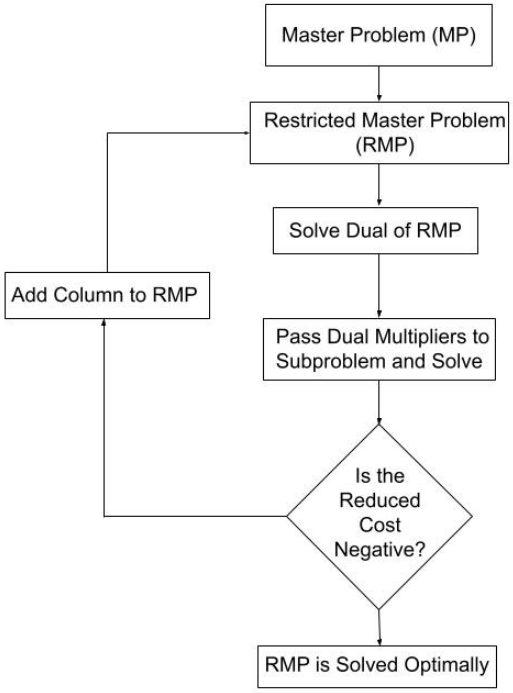


Figure 4.1. Framework of the column generation algorithm.

includes two pieces of information: the total load of the batch represented by the path, i.e. $c(p) = \sum_{o \in O} c_o a_{op}$ and the reduced cost of the batch, i.e. $\tilde{c}(p) = d_p - \sum_{o \in O} \mu_o a_{op}$, where d_p is the total distance of the batch, and $a_{op} = 1$ if and only if order o belongs to path p. A label associated with path p extends from node n to node n' for $n' > n$ if $c(p) + c_{n'} \leq C$ holds, where $c_{n'}$ is the number of items of the order at node n'. In that case, a new label corresponding to the path p' is created at node n' with a new total load $c(p') = c(p) + c_{n'}$ and a new reduced cost $\tilde{c}(p') = d_{p'} - \sum_{o \in O} \mu_o a_{op'}$ where $d_{p'}$ is calculated from scratch. After extending a label, two conditions are checked at the current node to remove some of the unpromising labels. A label associated with the path p at node n dominates another label associated with the path q at the same node if both $c(p) \leq c(q)$ and $\tilde{c}(p) \leq \tilde{c}(q)$ hold. When the dominance rule is applied and the dominated labels are removed, the MÖ labeling algorithm becomes a heuristic enumeration scheme because of the non-linear nature of the distance function.

If the new label corresponding to the path p' that is extended to node n' has a negative reduced cost, the corresponding batch is added to a batch pool in which

candidate batches are stored and added to the RMP. Otherwise, a lower bound, i.e. ω, on the reduced cost of the label is calculated considering only the orders that follow the current node n'. The set of nodes that follow node n' with positive dual values is denoted by $\overline{V} = \{n'' \in V : n'' > n' \wedge \mu_{n''} > 0\}$. If ω is negative, then the label is extended.

The labeling algorithm enumerates many batches with negative reduced cost, and the generated batches are stored in a pool. At the end of each iteration, only the batch with the most negative reduced cost is added to the RMP. After solving the RMP, the dual variables and the reduced costs of the batches in the pool are updated. In case there exists a batch with negative reduced cost, it is added to the RMP without running the labeling algorithm. Otherwise, when there is no batch with negative reduced cost in the pool, the labeling algorithm with heuristic enumeration starts. When the number of the batches stored in the pool reaches its limit, denoted by θ, a new batch is added to the pool only if its reduced cost is the minimum in the pool. Whenever the heuristic enumeration fails to generate a batch, the exact enumeration is executed and the pool is formed as explained previously.

The second labeling algorithm which we employ to solve the PSP is developed by Wahlen and Gschwind [37], where the PSP is defined as a shortest path problem with capacity constraints. Their procedure is referred to as the WG labeling algorithm where a graph similar to the one used in Muter and Öncan [35] is utilized. The nodes in the graph are sorted in such a way that $\mu_k/c_k > \mu_{k+1}/c_{k+1}$ holds, and each node is connected to the next one with two parallel arcs. If the label is extended through the first (second) arc, the order of the next node is included in (excluded from) the batch. Since any label ℓ with a non-negative lower bound on its reduced cost can be discarded, a completion bound of the labels should be calculated with respect to their current nodes and total loads. A completion bound is the amount of dual price that a label ℓ can obtain from the remaining nodes during the algorithm. For this purpose, a backward labeling algorithm is implemented from the sink to the source. Each label in the backward labeling algorithm, e, which represents a completion bound, is associated

with $(v(e), c(e), \pi(e))$ which are its first node, used capacity, and total dual price. A label e dominates another label e' on the same node if both of the constraints

$$c(e) \leq c(e'), \tag{4.7}$$

and

$$\pi(e) \geq \pi(e') \tag{4.8}$$

are satisfied.

At the end of the backward labeling algorithm, there is a set of non-dominated labels ε_n at each node n. The completion bound of a label ℓ on node n with total load c_ℓ is given as $\max_{e \in \varepsilon_n : c(e) \leq C - c(\ell)} \pi(e)$. Later, batches with negative reduced cost are generated via the forward labeling algorithm. A label is discarded if its reduced cost is greater than or equal to its completion bound. While in the MÖ labeling algorithm only the batch with the most negative reduced cost is added to the RMP, every batch with negative reduced cost is added to the RMP in the WG labeling algorithm. The algorithm ends when the number of labels with negative reduced cost at a node n reaches its capacity $\psi \cdot n$, where ψ is a constant. This is an acceleration technique, which turns the labeling algorithm into a heuristic approach. This approach will be referred as CCG-H. When CCG-H fails to find a column with negative reduced cost, the exact labeling algorithm has to be invoked ignoring the limit of the number of labels. Note that Wahlen and Gschwind [37] do not apply any dominance rule among the labels in the forward labeling phase.

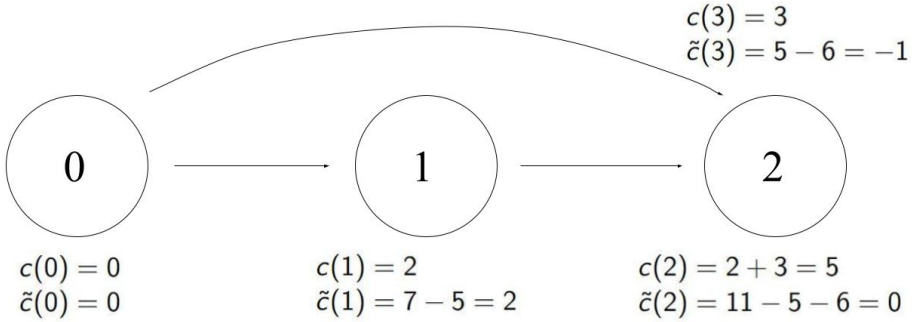
Both labeling algorithms are implemented in our CG procedure to compare their performances. The algorithms are outlined with Figure 4.2 for a small example with two orders $O = \{1,\ 2\}$ and batch capacity $C = 5$. The dual prices μ_1 and μ_2 are 5 and 6 with batch sizes, c_1 and c_2, of 2 and 3, respectively. It takes 7 and 5 units of distance to collect the orders separately while collecting them together has a distance of 11 units. Figure 4.2(a) (Figure 4.2(b)) represents the MÖ (WG) labeling algorithm.

4.5. Obtaining the Optimal Solution

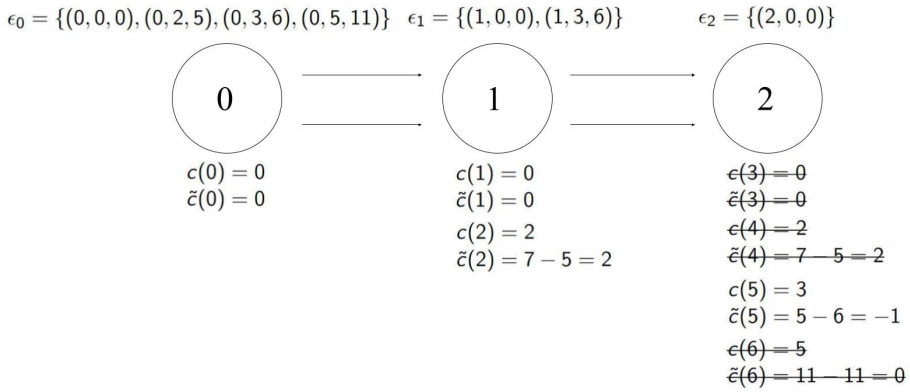
The column generation does not guarantee an integer solution and the existence of the optimal columns in the RMP. Baldacci et al. [52] propose an exact algorithm for the Capacitated Vehicle Routing Problem (CVRP). They first obtain lower and upper bounds for the problem, and then generate columns whose negative reduced cost are greater than $UB - LB$. This difference between the bounds is called the gap. The MP is then solved with an integer programming solver and the optimal solution is obtained.

This method is based on the proposition from the book "Integer and Combinatorial Optimization" by Wolsey and Nemhauser [53]. The proposition states that, quoting from their book: "If x_j is nonbasic at its lower (upper) bound in the solution of LP, $x_j \in Z^1_+$, and $z_{LP} + \bar{c}_j \leq \underline{z}(z_{LP} - \bar{c}_j \leq \underline{z})$, there exists an optimal solution to the integer program with x_j at its lower (upper) bound" (page 389, Proposition 2.1). They compare their approach with two other methods on the same problem; a BC algorithm and a BPC algorithm. Their algorithm outperforms BC in every instance they solve. BPC manages to solve 11 out of 11 instances to optimality while the proposed algorithm can only solve six of them, yet it reaches the optimal solution faster.

Baldacci et al. [54] develop the column-and-cut generation (CCG) method to solve the CVRP. They use the proposition of Wolsey and Nemhauser [53] in this study as well. To obtain better lower bounds they add some valid inequalities, namely, capacity constraints, set partitioning inequalities, framed capacity inequalities, comb inequalities and hypotour inequalities. Their lower bounds are reported to be better than those in the literature. This enables their algorithm to obtain the optimal solutions faster than the BPC algorithm, which is compared to their work on the instances of CVRP. Baldacci et al. [55] improve their CCG in [54] to solve the Capacitated Vehicle Routing Problem (CVRP) and the Vehicle Routing Problem with Time Windows (VRPTW).



(a) MÖ labeling algorithm.



(b) WG labeling algorithm.

Figure 4.2. Representation of the labeling algorithms.

Baldacci et al. [54] develop the column-and-cut generation (CCG) method to solve the CVRP. They use the proposition of Wolsey and Nemhauser [53] in this study as well. To obtain better lower bounds they add some valid inequalities, namely, capacity constraints, set partitioning inequalities, framed capacity inequalities, comb inequalities and hypotour inequalities. Their lower bounds are reported to be better than those in the literature. This enables their algorithm to obtain the optimal solutions faster than the BPC algorithm, which is compared to their work on the instances of CVRP. Baldacci et al. [55] improve their CCG in [54] to solve the Capacitated Vehicle Routing Problem (CVRP) and the Vehicle Routing Problem with Time Windows (VRPTW).

Glize et al. [56] propose a CG algorithm for the Bi-ObjectiveTeam Orienteering Problem with Time windows. At the end of column generation phase, they generate

the columns whose reduced costs are less than or equal to the gap. They compare their algorithm with the only other exact algorithm for this problem, which is a BB algorithm, and show that the CG algorithm outperforms the BB algorithm. Muter [57] proposes an exact algorithm to minimize the makespan on single and parallel batch processing machines. The author uses the same strategy at the end of the CCG. This algorithm is compared to the MIP formulations of the same problem, and proven to be more efficient.

Generating columns whose reduced costs are not greater than the gap can also be applied to the BP or BPC algorithms. Subramanyam et al. [58] solve the Range-Extended Electric Vehicle Routing Problem (REEVRP) by producing such columns when the gap is sufficiently small during the BPC algorithm. Pessoa et al. [59] present a BPC solver for a class of the VRP. The column generation phase generates routes whose reduced costs are less than the gap. Queiroga et al. [60] develop two Branch-Cut-and-Price (BCP) algorithms for the Vehicle Routing Problems with Backhouls (VRPB). They also generate the routes with reduced costs not greater than the gap when the gap is small enough to solve the RMP using a MIP solver. Another application of generating the columns with respect to the gap has been proposed by Damiao et al. [61]. Their BCP algorithm for the Cumulative Capacitated Vehicle Routing Problem (CCVRP) performs until the gap to get sufficiently small enough to enumerate such columns. They compare their work with another BCP algorithm from the literature for the same problem. Contardo and Martinelli [62] solve the Multi-Depot Vehicle Routing Problem (MDVRP). Their solution method is based on two formulations: a vehicle-flow formulation and a set partitioning formulation. They generate the columns of the set partitioning formulation based on the gap, and after solving the LP relaxation of the vehicle-flow formulation they remove the variables, which means removing the corresponding edges from the network, whose reduced costs are greater than the gap itself.

The same approach is also used in order batching related problems. Muter and Öncan [35] develop a CG algorithm and compare their results with another approach

which is based on enumerating all feasible batches. Since the latter requires extensive computational time, their work is proven to be more efficient, especially in larger instances. Muter and Öncan [36] solve the Order Batching and Picker Scheduling Problem, for a given routing policy. They minimize the travel time and the makespan of the pickers. The columns with negative reduced costs less than or equal to the gap are used at the end of their CG procedure.

Therefore, in our CG procedure, after adding the columns with negative reduced cost to the RMP, the RMP is solved and a LB (the LP relaxation) of the problem is obtained. Then the integrality constraints are added to the RMP to obtain an UB. Since the number of columns in RMP is relatively small, the MILP solution does not require long execution times. Using the both UB and LB values, the labeling algorithm is run once more to add the columns with reduced cost lower than $UB - LB$ to the RMP, and with the integrality constraints, the RMP is solved to obtain the optimal solution.

4.6. Branch-Price-and-Cut

The second solution approach is based on branching when the solution is fractional. We use two branching decisions considering the total number of pickers, and based on Ryan-Foster branching rule.

After the CG procedure, if the solution is not integer, a branch-and-bound tree is formed. In the first node, if the total number of pickers, $\sum_{b\in B}\sum_{r\in R}\bar{x}_r^b$ where $\bar{x}_r^b$ is the optimal solution to the RMP, is not integer, two branches are created by adding the constraints $\sum_{b\in B}\sum_{r\in R}x_r^b \leq \left\lfloor \sum_{b\in B}\sum_{r\in R}\bar{x}_r^b \right\rfloor$ and $\sum_{b\in B}\sum_{r\in R}x_r^b \geq \left\lceil \sum_{b\in B}\sum_{r\in R}\bar{x}_r^b \right\rceil$.

The second branching scheme that we use is the well-known Ryan-Foster branching rule. Let $f_{o_1o_2} = \sum_{b\in B}\sum_{r\in R}a_{o_1b}a_{o_2b}\bar{x}_r^b$ be the total number of times that orders $o_1, o_2 \in \mathcal{O}$ are in the same batch. For fractional values of $f_{o_1o_2}$, two branches are created. In the first (separate) branch, orders are assigned to separate batches by adding

the constraints $x_r^b \leq 0$ for which $a_{o_1 b} = a_{o_2 b} = 1$. In the second (together) branch, the orders are assigned to the same batch by adding the constraints $x_r^b \leq 0$ for which $a_{o_1 b} + a_{o_2 b} = 1$. Columns with $a_{o_1 b} = a_{o_2 b} = 0$ are allowed in both branches. Barnhart et al. [63] show that this branching rule guarantees an integer solution for SCP type formulations.

We consider 25 order pairs to branch on at the root node and this number is decreased by a factor of two at every level of the branch-and-bound tree. For each pair, the RMPs of two child nodes with corresponding branching constraints are solved in order to select the most promising branching decision. The decision is made by the product rule as proposed by Achterberg [64].

During the labeling algorithm, it is not allowed for a label to be extended to the next node if after this extension it will consist of two orders from a separate branch. If only one order from a together branch exists in a label, the corresponding batch does not enter the RMP, yet the label is allowed to be extended.

4.7. Generating Cuts

A well-known type of valid cuts is the subset row inequalities (SRIs) proposed by Jepsen et al. [65]. SRIs, which are utilized within several exact solution algorithms for OBP, which are used by [16, 35, 37, 57], are given as

$$\sum_{r \in \mathcal{R}} \sum_{b \in \mathcal{B}} \left\lfloor \frac{1}{\beta} \sum_{o \in \mathcal{S}} a_{ob} \right\rfloor x_r^b \leq \left\lfloor \frac{|\mathcal{S}|}{\beta} \right\rfloor \qquad \mathcal{S} \subseteq \mathcal{O}, \tag{4.9}$$

where $\mathcal{S}$ is a subset of orders and β is a parameter that takes values in the interval $2 \leq \beta \leq |\mathcal{S}|$. In our experiments we use the values adopted in [65] and [35], that is, $|\mathcal{S}| = 3$ and $\beta = 2$. This choice leads SRIs to take the form of

$$\sum_{r \in \mathcal{R}} \sum_{b \in \mathcal{B}} \left\lfloor \frac{1}{2} \sum_{o \in \mathcal{S}} a_{ob} \right\rfloor x_r^b \leq 1 \qquad \mathcal{S} \subseteq \mathcal{O} : |\mathcal{S}| = 3. \tag{4.10}$$

SRIs are enumerated explicitly at the end of the CG process and the violated ones are added into the RMP. To determine the effect of SRIs on reduced cost calculations, each

label ℓ contains additional information g_ℓ for each SRI, say g, which is the total number of orders $o \in \mathcal{S}$ that have the label ℓ. When extending the label ℓ at node n to label ℓ' at node n', the accumulated dual price $\pi(\ell')$ is given by $\pi(\ell) + \mu_{n'} + \sum_{g \in \mathcal{G}: g_{\ell'} \geq 2, g_\ell \leq 1} \sigma_g$, where σ_g is the dual price of the SRI g. Since the dual prices of SRIs are non-positive, the optimality of the labeling algorithm is not violated even if the effect of SRIs' dual prices on the lower bounds of the reduced cost values of batches is ignored.

For the Joint Order Batching and Picker Routing Problem, [14] improve the LP relaxation of the MP by imposing a lower bound on the number of pickers with a set of constraints, which are called capacity cuts (CCs). These cuts, which are also used in Wahlen and Gschwind [37], are given as

$$\sum_{r \in \mathcal{O}} \sum_{b \in \overline{\mathcal{B}}} x_r^b \geq \gamma(\mathcal{S}) \qquad \mathcal{S} \subseteq \mathcal{O}, \tag{4.11}$$

where $\gamma(\mathcal{S})$ is the minimum number of batches which is required to pick all orders in $\mathcal{S}$ and $\overline{\mathcal{B}} = \{b \in \mathcal{B} : b \cap \mathcal{S} \neq \emptyset\}$. The value of $\gamma(\mathcal{S})$ is calculated in the same way as described in [37] as

$$\gamma(\mathcal{S}) = \max \left\{ \left\lceil \frac{1}{C} \sum_{o \in \mathcal{S}} c_o \right\rceil, \left| o \in \mathcal{S} : c_o > \frac{C}{2} \right| + \left\lceil \frac{1}{C} \sum_{o \in \mathcal{S}: c_o = \frac{C}{2}} c_o \right\rceil \right\}. \tag{4.12}$$

The separation of the CCs for each routing policy $r \in \mathcal{R}$ is accomplished by means of two heuristics and an MIP-based approach. The cuts generated by these approaches will be referred to as CC1, CC2 and CC3. In the first heuristic (CC1), for a subset of orders $\mathcal{S}$, batches $b \in \overline{B}$ are sorted so that $x_r^b - \gamma(b) \geq x_r^{b+1} - \gamma(b+1)$ holds. At each iteration, the orders of the first batch b in the sorted list are removed from $\mathcal{S}$. Whenever the current subset of orders $\mathcal{S}$ violates inequality (4.11), the corresponding cut is added to the RMP. The heuristic stops when S is empty.

The second heuristic (CC2) starts with the set $\mathcal{S}$ that has a single order, and each order is tried as the starting point of the algorithm. At each iteration, a single order o is added to the set that maximizes the expression $m_o(\mathcal{S}) = \frac{\gamma(\mathcal{S} \cup \{o\}) - \gamma(\mathcal{S})}{\bar{x}_r^{\mathcal{S} \cup \{o\}} - \bar{x}_r^{\mathcal{S}}}$ where $\bar{x}_r^{\mathcal{S} \cup \{o\}}$ and $\bar{x}_r^{\mathcal{S}}$ are the optimal values of the corresponding decision variables. The algorithm stops when a violation of the capacity cut is detected.

For the MIP-based approach (CC3), we introduce a new set of parameters and variables. Let the binary decision variable y_o be equal to one if the order o is in the searched subset $\mathcal{S}$. $\lambda_r^b = 1$ if batch $b \in \mathcal{B}$ of the routing policy $r \in \mathcal{R}$ involves any order of subset $\mathcal{S}$. The integer variable K represents $\gamma(\mathcal{S})$. The continuous variable $f \in [0, 1)$ is the fractional difference between K and $\gamma(\mathcal{S})$.

$$\max K - \sum_{r \in \mathcal{R}} \sum_{b \in \mathcal{B}} \lambda_r^b \bar{x}_r^b, \tag{4.13}$$

subject to

$$K = f + \sum_{o \in \mathcal{O}} \frac{c_o y_o}{C}, \tag{4.14}$$

$$\lambda_r^b \leq \sum_{o \in b} y_o \qquad r \in \mathcal{R},\ b \in \mathcal{B}, \tag{4.15}$$

$$\mid b \mid \lambda_r^b \geq \sum_{o \in b} y_o \qquad r \in \mathcal{R},\ b \in \mathcal{B}, \tag{4.16}$$

$$\lambda_r^b \in \{0, 1\} \qquad r \in \mathcal{R},\ b \in \mathcal{B}, \tag{4.17}$$

$$y_o \in \{0, 1\} \qquad o \in \mathcal{O}, \tag{4.18}$$

$$0 \leq f \leq 1 - \epsilon, \tag{4.19}$$

$$K \in Z_+^0. \tag{4.20}$$

The objective function (4.13) maximizes the violation of CC (4.11). The constraint (4.14) calculates the value of K. The constraints (4.15) and (4.16) define the relationship between λ_r^b and y_o. The constraints (4.17), (4.18), (4.19), and (4.20) are the domain definitions of the decision variables.

Adding CCs to the RMP affects the labeling algorithm since the reduced costs and lower bounds of the labels are influenced by the dual prices of the newly added CCs. Let $\mathcal{T}$ be the set of CCs. The reduced cost of a batch b is given by $d_r^b - \sum_{o \in b} \mu_o u_o - \sum_{t \in \mathcal{T}: b \in \overline{B}} \rho_t$ where ρ_t is the dual price of the cut t. To this end, each label $\ell(e)$ contains additional information for each CC say $t \in \mathcal{T}$, $t_\ell(t_e)$, which is the total number of orders $o \in t$ on the label $\ell(e)$. When extending the label ℓ with dual price $\pi(\ell)$ at node n to ℓ' at node n', the accumulated dual price $\pi(\ell')$ is given by $\pi(\ell) + \mu_{n'} + \sum_{t \in \mathcal{T}: t_{\ell'} \geq 1 \wedge t_\ell = 0} \rho_t$.

After the addition of the CCs, the dominance relationship among the labels, which are used in the backward labeling algorithm, must be satisfied. For a label e to dominate another label e' at the same node, inequality (4.8) must be updated as

$$\pi(e) - \sum_{t \in \mathcal{T}: t_e \geq 1, t_{e'} = 0} \rho_t \geq \pi(e'). \tag{4.21}$$

The completion bound of a label ℓ with total load $c(\ell)$ at node n is given as

$$\max_{e \in \varepsilon_n : c(e) \leq C - c(\ell)} \pi(e) + \sum_{t \in \mathcal{T}: t_e \geq 1, t_\ell = 0} \rho_t, \tag{4.22}$$

where ε_n is the set of nondominated backward labels at node n, and $n(o)$ is the order associated with node n.

5. EXPERIMENTS AND RESULTS OF JOBRPSP

5.1. Generation of the First Test Instances

We use the instances of Muter and Öncan [35] for computational experiments (INS_{MO}), where the warehouse is assumed to have 10 aisles and 10 storage locations on both sides of each aisle, as can be seen in Figure 3.1. As the distance between consecutive storage locations is 1 m, the length of each aisle is $L = 12$ m, and the distance between back (front) aisle and the nearest storage location is 1.5 m. The width between two neighboring picking aisles is $w = 2.4$ m. The depot is located in front of the leftmost (first) aisle. The number of orders $|\mathcal{O}|$ takes values from the set $\{20, 30, ..., 100\}$ and three different values are used for the batch capacity C where $C = \{24, 36, 48\}$. For each number of orders and batch capacity, 10 instances are generated. The number of items in each order varies uniformly between 2 and 10, and the volume of each item is taken as one. This implies that the order volume c_o varies uniformly between 2 and 10. Since the instances in Muter and Öncan [35] only use the random storage policy, we generate item classes for the other storage policies considered, i.e., within-aisle, perimeter, across-isle, and diagonal policies, and locate the items to the storage locations accordingly. The algorithms are coded in C++ within the Visual Studio 2022 environment, and Cplex 12.10.0 is used to solve MILPs and LPs. All experiments are performed in a platform with an Intel(R) Core(TM) i5-6500 CPU @ 3.20 GHz processor and 8 GB Ram.

5.2. Comparison of the Labeling Algorithms

Before we carry out computational experiments to assess the effectiveness and efficiency of the CCG and BPC methods, we compare the performance of the labeling algorithms used in these methods. To this end, we solve the JOBRPSP with batch capacity $C = 24$ and the order sizes $|\mathcal{O}| = \{20, 30, ..., 100\}$ for CCG and $|\mathcal{O}| = \{20, 30, ..., 80\}$ for BPC under both random storage policy and perimeter storage policy. In total,

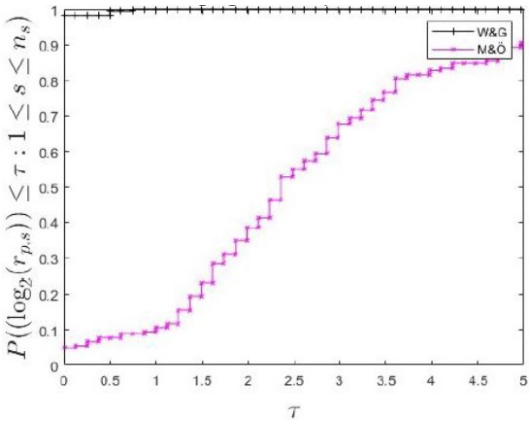
there are 180 instances for the comparison of the labeling algorithms. The time limit for both algorithms is set to 3600 seconds.

The results of 180 instances are reported with Table 5.1. The first column of CCG represents the gap computed as $(UB - LB)/LB$. The second column includes the percentage of optimally solved instances, while the average CPU time in seconds over the instances is shown in the third column. The superiority of the WG algorithm is evident in terms of CPU time and gap values. The advantage of the WG labeling algorithm lies in the fact that it adds, at every iteration, all columns with a negative reduced cost to the RMP, while the MÖ algorithm adds only the column with the most negative reduced cost. This implies that a larger RMP needs to be solved by the WG algorithm, which is then converted into an MILP model to obtain an UB. Thus, the UB and the gap from the WG algorithm are smaller in more than half of the instances. The WG algorithm performs better with BPC as well.

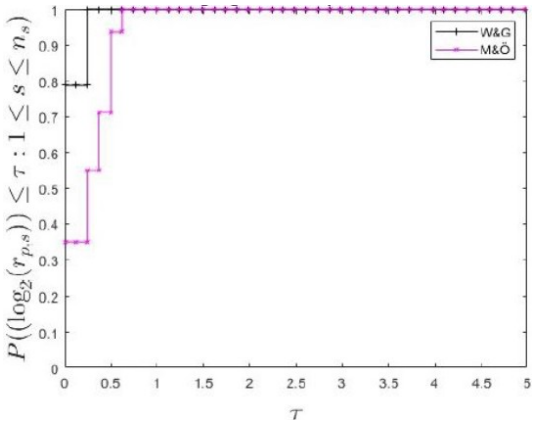
Table 5.1. Comparison of the labeling algorithms.

	CCG			BPC	
Algorithm	Gap	Opt. solved (%)	CPU time (s)	Opt. solved (%)	CPU time (s)
WG	0.74	97	219.32	84	894.94
MÖ	1.65	79	920.71	82	951.31

To better emphasize the performance difference between the two labeling algorithms, we refer to the work by Dolan and More [66], and draw the performance profiles of the algorithms in Figure 5.1. It can be easily seen that the WG algorithm outperforms the MÖ algorithm for both CCG (Figure 5.1(a)) and BPC (Figure 5.1(b)) approaches. As a result, we opt to employ the former labeling algorithm in our experiments.



(a) Labeling Algorithm Comparison for CCG.



(b) Labeling Algorithm Comparison for BPC.

Figure 5.1. Performance profile of the labeling algorithms on [0, 5].

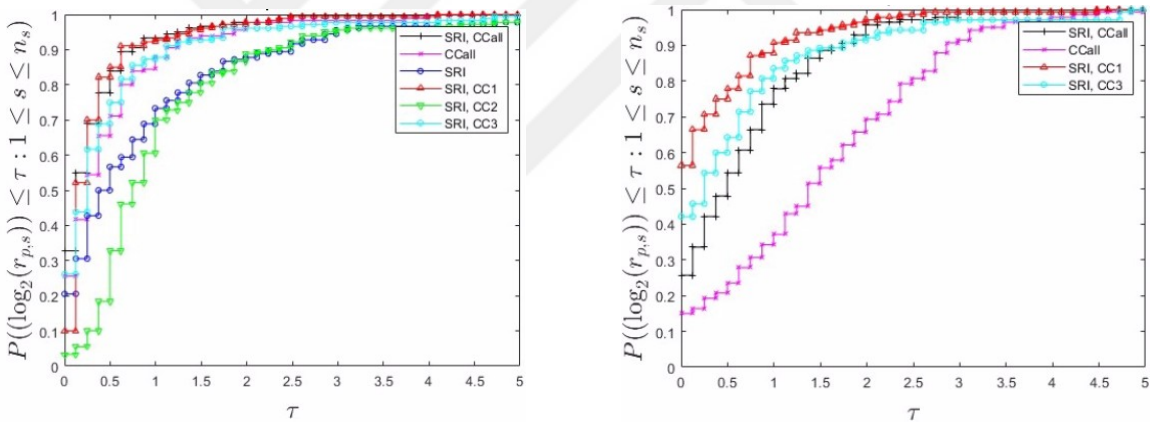
5.3. Measuring the Effect of the Cuts

To investigate the effect of the two types of the cuts, i.e., SRIs and CCs, we solve the same instances by incorporating these cuts separately (SRI and CCall) and together (SRI+CCall). Table 5.2 presents the results for the same instances that are used in the comparison of the labeling algorithms. We also investigate the effect of three classes of CCs when used separately (SRI+CC1, SRI+CC2, and SRI+CC3) or together (SRI+CCall). We compare six variants of CCG and four variants of BPC since SRI and SRI+CC2 perform poorly even in small instances. The first column of both algorithms indicates the percentage of optimally solved instances, and the second column is the average CPU time of the entire algorithm in seconds. As can be observed in Table 5.2, CCs seem to be more crucial than SRIs in terms of reducing the CPU time and increasing the number of optimally solved instances.

Using SRIs in addition to CCs turns out to be the best version of CCG, while using only CC1 in addition to the SRIs outperforms other variants for BPC. Figure 5.2 depicts the performance profiles of the variants for both algorithms. Figure 5.2(a) (Figure 5.2(b)) represents the CCG (BPC) algorithm.

Table 5.2. The effect of cuts.

	CCG		BPC	
Added cuts	Opt. solved (%)	CPU time (s)	Opt. solved (%)	CPU time (s)
SRI+CCall	96.7	219.32	82	900.39
CCall	95.0	277.03	69	1332.32
SRI	93.9	354.91	-	-
SRI+CC1	96.7	222.59	84	894.94
SRI+CC2	95.6	364.66	-	-
SRI+CC3	96.7	219.32	81	944.85



(a) Cut Comparison for CCG. (b) Cut Comparison for BPC.

Figure 5.2. Performance profile of cuts on [0, 5].

5.4. Comparison of CCG and BPC

Next, we compare the two solution methods for five storage policies. We solve instances with $C = 24$, and with number of orders up to 100 for with CCG, and orders up to 80 orders with BPC since larger instances cause out of memory error. This shows the superiority of CCG alone, yet we also provide the average execution times and percentages of optimally solved instances for both methods, for instances with number of orders up to 80. Table 5.3 incorporates the results obtained with two

methods. CCG outperforms BPC in both execution time and percentage of optimally solved instances. Figure 5.3 illustrates the performance profiles of both methods.

Table 5.3. Comparison of CCG and BPC.

	CCG		BPC	
Storage Policy	CPU time (s)	Opt. solved (%)	CPU time (s)	Opt. solved (%)
Random	80.68	99	886.27	83
Within	202.34	97	1578.54	64
Perimeter	103.58	99	903.61	84
Across	36.81	100	1129.21	79
Diagonal	100.4	100	1311.33	70

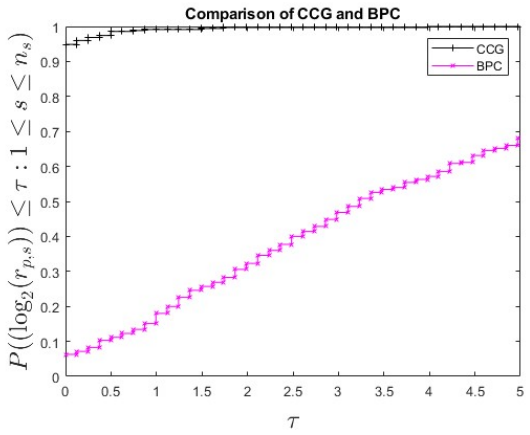


Figure 5.3. Performance profile of CCG and BPC on [0, 5].

5.5. Complete Enumeration

A complete enumeration procedure can also be used to solve the JOBRPSP with a given storage policy where all feasible batches are enumerated and incorporated into the complete (or unrestricted) MP, which is then solved by an MILP solver. The results of this approach are provided in Table 5.4 for the same instances used before for the

comparison of the labeling algorithms and the evaluation of the benefit of the cuts. Due to the memory limitation of the computer, we can only generate 700,000 batches (columns) during the solution procedure.

Table 5.4. Results of the complete enumeration procedure.

# orders	CPU time (s)	# Opt. solved
20	2.39	10
30	770.58	9
40	1897.40	6

The total CPU time required for the complete enumeration procedure is significantly higher than that of the CCG method since the average number of batches for the instances with 40 orders is 334,769, while the CCG method has only 49,362 columns at the last iteration when the MP is solved to optimality.

5.6. Determination of the Most Selected Routing Policies

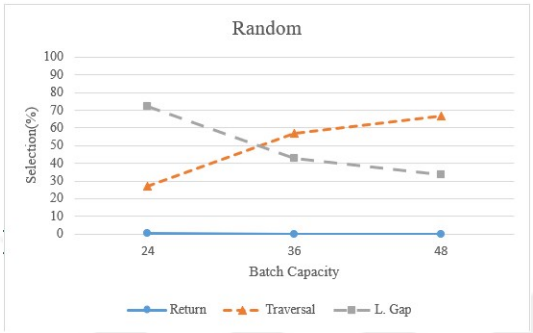
After all the components of the CCG method have been identified, it is now possible to solve JOBRPSP. We solve the instances under each storage policy by taking into account the routing selection aspect, i.e., the JOBRPSP. The results are included with Table 5.5. The numbers in the columns associated with the routing policies represent the percentage of the batches collected using that routing policy. The column titled "Average Objective Value" denotes the average distance traveled for each batch capacity and storage policy. We can solve all the instances when $C = 24$, whereas we can solve instances up to 40 orders when $C = 36$ and up to 10 orders when $C = 48$. The last column contains the average computation time spent to solve these instances.

Table 5.5. Selection percentages of routing policies for INS_{MO}.

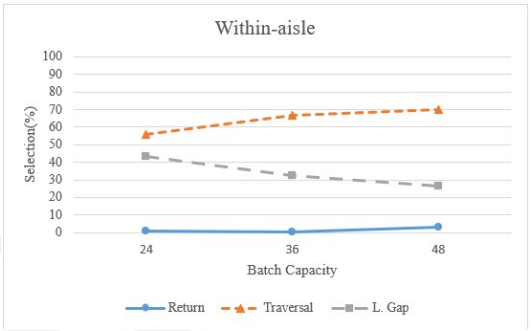
		Routing Policy				
Storage P.	C	Return	Traversal	L. Gap	Avg. Obj. Val.	CPU time (s)
Random	24	1	27	72	1788.09	103.39
	36	0	57	43	714.37	406.55
	48	0	67	33	398.12	1072.48
Within	24	1	56	43	990.83	337.14
	36	1	67	33	329.03	624.78
	48	3	70	27	169.2	1275.80
Perimeter	24	1	1	98	1443.52	149.17
	36	1	0	99	562.2	84.76
	48	3	0	97	331.29	1324.64
Across	24	98	0	2	1441.02	104.58
	36	100	0	0	543.47	778.99
	48	100	0	0	297.24	979.53
Diagonal	24	10	65	26	1077.34	207.68
	36	7	83	10	369.09	459.17
	48	7	90	3	191.4	285.26

A quick inspection of the results indicates that the largest gap and traversal routing policies are the preferred routing policies in terms of the routes selected. The largest gap routing policy is the most selected for the perimeter storage policy at all batch sizes and for the random storage policy when the batch capacity has its smallest value. The traversal routing policy ranks the first for within-aisle and diagonal storage policies, and for the random storage policy at larger batch capacities. It is also evident that a given routing policy can perform well for particular storage policies. For example, the return routing policy ranks first when the across-aisle storage is adopted. The performance of the routing policies is also illustrated with Figure 5.4, where for each storage policy the proportions of the selected routing policies are plotted. Selection

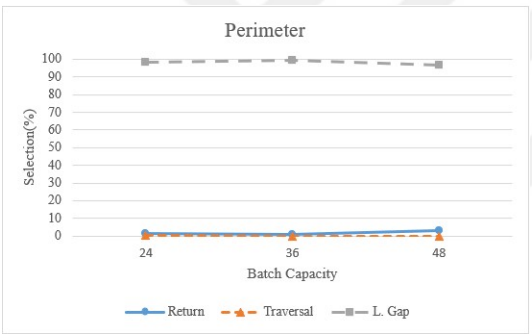
percentages of the routing policies when random, within-aisle, perimeter, across-aisle, and diagonal storage policies are given with Figure 5.4(a), Figure 5.4(b), Figure 5.4(c), Figure 5.4(d), and Figure 5.4(e), respectively.



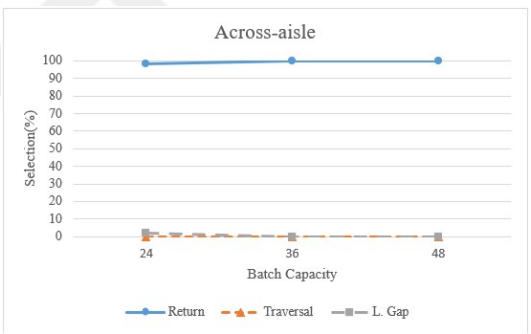
(a) Random



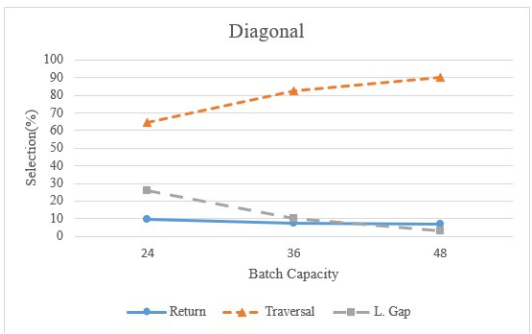
(b) Within-aisle



(c) Perimeter



(d) Across-aisle



(e) Diagonal

Figure 5.4. Routing heuristic and storage policy interaction.

The results reveal that the most frequently selected routing policy over all storage policies is the traversal, while the largest gap routing policy is the runner-up. It is interesting to note that the return routing policy is almost never selected when the

random storage policy is used. For this particular storage policy, it was previously mentioned that the traversal routing policy is the most selected policy except at the smallest batch capacity. It appears that as batch capacity increases, the selection rate of this routing policy increases because it makes more sense to travel the entire picking aisle when there are more items to collect. This result coincides with the finding of Shqair et al. [31] which states that as the number of pick locations in a batch increases, traversal routing becomes more efficient. Our results show that for random, within-aisle and diagonal storage policies, the batch capacity increases along with the selection percentage of traversal routing policy.

A byproduct of the computational results is to validate the conclusions drawn by other researchers in the framework of order batching, picker routing, and storage policy selection. In their paper, Bahçeci and Öncan [18] show that the largest gap is the best routing policy for the random storage policy, while the rest of the routing policies except return routing policy have close results to the largest gap in terms of the total distance of the batches. The plot of the random storage policy described with Figure 5.4 also justifies the superiority of the largest gap routing policy and the inferiority of the return routing policy. Frankze et al. [32] report that the traversal routing policy outperforms the return and largest gap routing policies when within-aisle or diagonal storage policies are used. The plots for the within-aisle and diagonal storage policies illustrated with Figure 5.4 are in alignment with this conclusion.

In their paper, Petersen and Schmenner [23] investigate the relationship between routing and storage policies by performing experiments with six routing policies (return, traversal, midpoint, largest gap, composite, and optimal) four storage policies, and two depot locations. The authors claim that largest gap is the best routing policy for the perimeter storage policy in minimizing the total distance traveled. Our results support this claim since the largest gap turns out to be the most preferred routing policy.

Caron et al. [22] compare traversal and return routing policies when across-aisle storage policy is used, and assert that the return routing policy should be preferred.

Indeed, our results favor this finding as the return routing clearly dominates the traversal routing. This result is not surprising because Class A items are located closer to the front aisle in across-aisle storage policy which favors the return routing policy.

5.7. Quantifying the Benefit of Using Multiple Routing Policies

Recall that one of the main objectives of this study is to assess the benefit of picker routing selection in comparison with using a single routing policy within the framework of OBP. Note that all the studies in the literature adopt a single routing policy. In this section, we solve JOBRPSP instances by assuming that a single-best routing policy is adopted. The average of optimal objective values computed single routing policy over all the instances are then compared with the optimal objective values obtained when multiple routing policies are adopted. We report the benefit of using multiple routing policies for each storage policy and batch capacity in Table 5.6, where the benefit of multiple routing is measured as a percent reduction in the traveling distance computed as $100 \times (Z_{\text{single}} - Z_{\text{multi}})/Z_{\text{single}}$. We also solve the same instances with optimal routing so as to compute the gap between optimal routing and multiple routing. In Table 5.6, the gap is computed as $100 \times (Z_{\text{multi}} - Z_{\text{optimal}})/Z_{\text{multi}}$.

As can be observed from Table 5.6, the benefit of using a multiple routing policy varies based on the storage policy and batch capacity. The benefit measured as the reduction in the traveling distance varies between 0% and 37.63%. The largest reduction of 37.63% is attained for the perimeter storage policy, against return routing policy when the batch capacity $C = 48$. The next analysis is related to determining the best storage policy given that the orders in different batches can be retrieved considering different routing policies, i.e., when routing selection is available. As stated above, Table 5.6 includes for each storage policy and batch capacity the average objective values for instances with multiple routing at three different batch capacity values. The within-aisle storage policy performs the best for all batch capacity values, while the diagonal storage policy comes second. Interestingly, Petersen and Schmenner [23] also claim that among the five storage policies considered in this paper, the within-aisle

and diagonal policies are the best when only one routing policy is used for all batches. Our results provide evidence for the same conclusion for the case of multiple routing and optimal routing.

Table 5.6. Comparison of objective values of different routing strategies for INS_{MO}.

			Gap			
Storage P.	C	Multiple R.	Return	Traversal	Largest Gap	Optimal R.
Random		1788.09	21.04	4.98	0.94	4.65
Within		990.8	23.07	2.88	3.89	3.19
Perimeter	24	1443.52	33.44	22.96	0.11	1.45
Across		1441	0	21.25	7.66	0.17
Diagonal		1077.34	13.96	2.17	5.81	5.14
Average		1348.16	18.3	10.85	3.68	2.92
Random		714.37	21.65	1.56	2.69	5.26
Within		329.03	25.18	2.06	6.52	2.6
Perimeter	36	562.2	36.62	23.22	0	1.06
Across		543.47	0	7.79	23.89	0.01
Diagonal		369.1	15.35	0.53	8.83	3.83
Average		503.63	19.76	7.03	8.39	2.55
Random		418.96	23.24	1.32	5.07	5.23
Within		169.2	26.65	1.03	10.17	2.84
Perimeter	48	331.29	37.63	20.42	0	1.55
Across		297.24	0	27.45	9.42	0
Diagonal		191.4	15.7	0.57	12.13	3.53
Average		281.62	20.64	10.16	7.36	2.63

5.8. Generation of the Second Set of Test Instances

As a second set of test instances, we use the instances generated by Van Gils et al. [48] (INS_{VG}). The same instances were also used for the same problem by Briant et al. [49]. The instances are divided into two major groups, which are small instances made of 6, 12 and 18 orders, and large instances with 100, 200 and 300 orders. In our experiments, we test our algorithm on all of the small instances and large instances with 100 orders, and trolley capacity of 15 and 30 items. Three storage policies are used in these instances, namely; random, within-aisle and across-aisle storage policies. A two-block warehouse layout is used.

5.9. Most Selected Routing Policies for the Second Set of Instances

The second set of instances, INS_{VG}, is solved via CCG. The average objective function values, CPU times and the frequencies of the routing policy selections for the small and large instances of INS_{VG} are reported with Table 5.7 and Table 5.8.

Table 5.7. Selection percentages of routing policies for small instances of INS_{VG}.

		Routing Policy				
Storage P.	C	Return	Traversal	L. Gap	Avg. Obj. Val.	CPU time(s)
Random	4	40	23	37	120,750.89	0.17
	8	27	15	58	96,643.11	2.40
	12	20	12	68	88,000.89	17.88
Within	4	35	30	35	111,453.89	0.14
	8	20	23	56	82,006.78	2.35
	12	18	20	62	76,703.78	9.50
Across	4	71	19	10	109,406.33	0.18
	8	77	7	16	81,838.78	3.51
	12	76	3	21	69,364.44	64.58

As expected, return routing policy dominates the others when across-aisle storage policy is used. For random and within-aisle storage policies, largest gap seems to be the best routing policy, especially when the batch capacity increases. These results overlap with the findings of INS_{MO}.

Table 5.8. Selection percentages of routing policies for large instances of INS_{VG}.

			Routing Policy				
Storage P	Aisle	C	Return	Traversal	L. Gap	Avg. Obj. Val.	CPU time(s)
Random	12	15	8	18	74	353,480.7	247.04
		30	2	52	46	290,716	80.97
	24	15	8	6	86	546,532	135.12
		30	1	6	93	440,634	84.82
	36	15	6	1	93	674,953.7	745.73
		30	4	0	96	575,381.3	76.78
Within	12	15	9	27	64	285,932.5	119.73
		30	5	33	62	212,083.7	45.07
	24	15	10	15	76	484,109	70.92
		30	1	25	73	353,166	37.76
	36	15	8	6	86	642,054.3	88.59
		30	5	18	77	532,482	39.43
Across	12	15	83	3	14	317,619.3	402.96
		30	72	4	24	242,255	38.92
	24	15	80	1	18	482,752.7	135.48
		30	75	0	25	391,499.7	130.35
	36	15	80	0	20	648,467.3	456.4
		30	62	1	36	536,344	137.69

5.10. Quantifying the Benefit of Using Multiple Routing Policies for the Second Set of Instances

The benefit of using multiple routing policies against single routing policy for small and large instances of INS_{VG} are analyzed and presented with Table 5.10 and Table 5.9, respectively. For small instances, usage of multiple routing policies can save up to 22.26% of the total picking time. This ratio increases to 29.27% when large instances are considered.

Table 5.9. Comparison of objective values of different routing strategies for large instances of INS_{VG}.

			Gap			
Storage P.	C	Multiple R.	Return	Traversal	Largest Gap	Optimal R.
Random		513,973.9	18.80	13.52	0.53	9.47
Within	15	493,794.4	17.68	10.07	0.73	8.18
Across		482,946.4	1.53	27.58	9.23	7.20
Average		496,904.9	12.67	17.06	3.50	8.28
Random		435,577.1	23.53	11.49	0.44	14.63
Within	30	365,910.6	20.59	7.66	0.72	7.32
Across		390,032.9	2.73	29.27	5.55	7.61
Average		397,173.5	15.62	16.14	2.23	9.85

For the two-block warehouse layout of INS_{VG}, across-aisle storage policy performs the best among three storage policies. With a relatively large cross-aisle width, and the most frequently used items located close to the first cross aisle of the first block, using multiple routing, single routing (return routing policy) or optimal routing when across-aisle storage policy is used, the total picking time is the lowest among other combinations of routing and storage policies.

Both set of instances use single depot. Layouts with multiple depots can also be considered. Xie et al. [4] state that multi-depot can bring savings in the total distance of the pickers especially when the items are not located in a single storage location but instead scattered across multiple storage locations in the warehouse.

Table 5.10. Comparison of objective values of different routing strategies for small instances of INS_{VG}.

			Gap			
Storage P.	C	Multiple R.	Return	Traversal	Largest Gap	Optimal R.
Random		120,750.89	6.54	5.57	2.92	1.12
Within	4	111,453.89	4.54	3.92	2.63	1.15
Across		109,406.33	0.90	9.90	8.79	0.91
Average		113,870.37	4.08	6.42	4.71	1.06
Random		96,643.11	11.48	8.48	1.54	5.66
Within	8	82,006.78	10.57	6.19	1.39	4.02
Across		81,838.78	1.81	16.10	10.12	2.63
Average		86,829.56	8.16	10.15	4.19	4.18
Random		88,000.89	15.12	11.71	1.22	7.95
Within	12	76,703.78	13.15	8.03	1.21	6.57
Across		69,364.44	2.11	22.26	9.39	3.75
Average		78,023.04	10.62	13.63	3.64	6.23

6. JOINT ORDER BATCHING, BATCH SEQUENCING AND PICKER ROUTING PROBLEM WITH DEADLINES

6.1. Problem Description

The set of orders, $\mathcal{O}$, is predefined. Each order consists of multiple items, and each item is stored at a distinct location within the warehouse. All orders must be fully collected, and splitting them is not allowed. Pickers use the optimal routing policy. For an order o, its size and deadline are represented by c_o and $\bar{d}_o$. The deadline indicates the latest possible time the order must arrive at the depot. The cardinality of the set of deadlines, $|\mathcal{D}|$, is typically smaller than $|\mathcal{O}|$ because several orders can share the same deadline.

$\mathcal{P}$ denotes the set of pickers. Pickers collect items by forming a batch—i.e., a group of orders—and follow a route $k \in \mathcal{K}$ to pick those items. The set of orders assigned to route k is denoted by $\mathcal{O}_k$. The total size of a batch must not exceed the trolley capacity C, ensuring that $\sum_{o \in O_k} c_o \leq C$. The deadline $\bar{d}_o$ for route k is defined as the earliest (i.e., most urgent) deadline among all orders in $\mathcal{O}_k$, formally: $\bar{d}_k = \min_{o \in O_k} \bar{d}_o$.

The parameter d_k represents the total time required for a picker to complete route k. It consists of three components: d_{setup}, a fixed setup time; d_{search}, the travel time needed to reach the item locations; and $d_{picking}$, the picking time, which depends on the total batch size.

Joint Order Batching, Picker Routing and Scheduling Problem with Deadlines (JOBPRSP-D) addresses the order batching problem with the goal of minimizing the overall time to collect all items while meeting their respective deadlines. We implement the CCG algorithm to deal with this problem.

6.2. Master Problem

The decision variable x_k^p is equal to 1 if and only if route k is performed by picker p. We denote the set of routes retrieving order o as $\mathcal{K}(o) = \{k \in \mathcal{K} : o \in \mathcal{O}_k\}$ and set of routes with deadline lower than or equal to $\bar{d}$ as $\mathcal{K}(\bar{d}) = \{k \in \mathcal{K} : \bar{d}_k \leq \bar{d}\}$. In the case of the JOBPRSP-D, the MP is given as

$$\min \sum_{k \in \mathcal{K}} d_k \sum_{p \in \mathcal{P}} x_k^p, \tag{6.1}$$

subject to

$$\sum_{k \in \mathcal{K}(o)} \sum_{p \in \mathcal{P}} x_k^p \geq 1 \qquad o \in \mathcal{O}, \tag{6.2}$$

$$\sum_{k \in \mathcal{K}(\bar{d})} d_k x_k^p \leq \bar{d} \qquad \bar{d} \in \mathcal{D},\ p \in \mathcal{P}, \tag{6.3}$$

$$x_k^p \in \{0, 1\} \qquad k \in \mathcal{K},\ p \in \mathcal{P}. \tag{6.4}$$

The objective function (6.1) seeks to minimize the total time required to complete all routes. Constraints (6.2) guarantee that each order is assigned for picking, while constraints (6.3) ensure that the deadline for each order is respected. Constraints (6.4) specify the domain restrictions of the decision variables. If all possible routes are generated, the MP can directly determine the optimal solution to the JOBPRSP-D. However, generating all possible routes becomes infeasible for large instances. Therefore, a Restricted Master Problem (RMP) is formulated, considering only a subset of routes $\mathcal{K}' \subseteq \mathcal{K}$. In the RMP, the integrality constraints (6.4) on the decision variables are also relaxed. Initially, $\mathcal{K}'$ may not contain the optimal columns, so the objective is to identify and add such columns to $\mathcal{K}'$, thereby enriching the RMP.

6.3. Modified Labeling Algorithm

We perform the same procedures as WG labeling algorithm in Chapter 4 with small modifications. When a label l corresponding to a route is created, at first the total distance of the label d_l is calculated. Let $\mathcal{D}_{O^+}$ be the set of deadlines of orders included within label l. If $d_l > \min\{\mathcal{D}_{O^+}\}$, label l is discarded.

Let $\xi_{\bar{d}}^{p}$ be the dual variable of the constraint set (6.3). When the label l is extended to a new node, it has a reduced cost $\tilde{c}(l) = d_l - \sum_{o\in\mathcal{O}(\ell')}\mu_o + \sum_{\bar{d}\in\mathcal{D}:d_{\ell'}\leq\bar{d}} d_{\ell'}\xi_{\bar{d}}^{p}$. $\xi_{\bar{d}}^{p}$ is not added to the backward labeling algorithm. It takes non-positive values and therefore does not affect the optimality of the backward labeling algorithm.

6.4. Martello and Toth's Cuts

In addition to SRIs, and CCs, Martello and Toth's Cuts (MTC) by Briant et al. [49] are also considered for the JOBPRSP-D. They define the set

$$\mathcal{Q}(\bar{d}) = \left\{ q \in \mathbf{N} | q \geq 1, q \leq \left\lfloor \frac{1}{2}\bar{d} \right\rfloor \right\}. \tag{6.5}$$

For a deadline $\bar{d} \in \mathcal{D}$ and an integer $q \in \mathcal{Q}(\bar{d})$, the following sets are denoted: $\mathcal{K}_1(\bar{d}, q) = \{k \in \mathcal{K}(\bar{d}) | d_k > \bar{d} - q\}$ and $\mathcal{K}_2(\bar{d}, q) = \{k \in \mathcal{K}(\bar{d}) | q \leq d_k \leq \bar{d} - q\}$. A MTC is given as

$$\sum_{p\in\mathcal{P}}\sum_{k\in\mathcal{K}_1(\bar{d},q)} \bar{d}x_k^p + \sum_{p\in\mathcal{P}}\sum_{k\in\mathcal{K}_2(\bar{d},q)} d_k x_k^p \leq \bar{d}|\mathcal{P}| \qquad \bar{d}\in\mathcal{D},\ q\in\mathcal{Q}(\bar{d}). \tag{6.6}$$

As proposed by [49], for each $\bar{d} \in \mathcal{D}$, we set $q = \bar{d} = d_k + 1$ in such a way that there exists a route $k \in \mathcal{K}(\bar{d})$ with $x_k^{p*} > 0$ and satisfying $d_k \geq \frac{1}{2}\bar{d} + 1$, where x_k^{p*} represents the optimal value of the decision variable x_k^p in the RMP. Let $\upsilon_q^{\bar{d}}$ denote the dual price associated with constraints (6.6). Dual price of a label ℓ, denoted by $\pi(\ell)$, is given as

$$\pi(\ell) = \pi(\ell) + \sum_{q\in\mathcal{Q}(\bar{d})} \left\{ \sum_{\bar{d}\in\mathcal{D}:d_\ell\leq\bar{d},d_\ell<\bar{d}-q} \bar{d}\upsilon_q^{\bar{d}} + \sum_{\bar{d}\in\mathcal{D}:d_\ell\leq\bar{d},q\leq d_\ell\leq\bar{d}-q} d_\ell\upsilon_q^{\bar{d}} \right\}. \tag{6.7}$$

7. EXPERIMENTS AND RESULTS OF JOBBSPRP-D

In this section we provide the results of our experiments on INS_{VG} regarding the effects of the valid inequalities and the performance of our results with previous work in the literature with respect to optimal solutions and execution times.

7.1. Generation of New Test Instances

For the JOBPRSP-D, we use the instances INS_{VG}. For this problem, we consider the deadline values of orders given in these instances.

As is the case in Van Gils et al. [48] and Briant et al. [49], optimal routing policy is employed in this study. For multiple-block warehouse layouts, the optimal routing can be found by an improved version of Ratliff and Rosenthall [51]'s algorithm. Pansart et al. [67] and Cambazard and Catusse [68] share the details of this algorithm in their work.

7.2. Results of the Small Instances and the Effect of Valid Cuts

We evaluate the impact of valid cuts on small instances by testing eight subsets, each consisting of three valid cuts, added to the RMP. The corresponding execution times (in seconds) are included with Table 7.1. All RMP variants successfully find the optimal solution across all instances. Although no substantial performance differences are observed among the versions, incorporating all valid cuts yields marginally better results across all order sizes. The performance of our CCG approach is quite close—though slightly inferior—to the complete enumeration method of Briant et al. [49], who report average execution times of 0, 1.5, and 49.4 seconds for order sizes of 6, 12, and 18, respectively.

Table 7.1. Execution times in seconds of small instances with different valid cuts.

$\lvert\mathcal{O}\rvert$	no cut	SRI	MTC	CC	no SRI	no CC	no MTC	all cuts
6	0.14	0.14	0.14	0.14	0.16	0.14	0.13	0.13
12	3.83	3.86	3.81	3.70	3.94	3.86	3.65	3.60
18	71.98	73.13	71.33	67.40	67.62	72.72	67.83	65.96
average	25.21	25.60	24.99	23.83	23.73	25.47	23.69	23.06

7.3. Results on the Large Instances

To address large instances, we employ the RMP variant that includes all valid cuts. Our results are compared against the CGH approach proposed by Briant et al. [49]. Table 7.2 incorporates the upper bounds, execution times in seconds, and optimality gaps for both the CGH of Briant et al. [49] and our CCG algorithm, applied to large instances with order size $|O| = 100$ and trolley capacity $C = 15$.

Our CCG algorithm successfully solves 24 out of 26 instances to optimality. The two remaining instances are solved to near-optimality, with optimality gaps below 0.2% by the end of the computation. The largest optimality gap belongs to instance 36. The labeling algorithm generates more than 1.6 million routes for this instance. The second highest number of columns belong to instance 169 with 100,000 routes. In 12 cases, our method produces superior upper bounds compared to Briant et al. [49], and does so in less CPU time—except for instance 169, which is highlighted in bold as an exception. For 13 instances, both algorithms yield the same upper bound; however, our CCG approach achieves these results with lower execution times. Therefore, we conclude that the CCG algorithm outperforms the method of Briant et al. [49] in terms of both solution quality and computational efficiency.

Table 7.2. Results on large instances with trolley capacity $C = 15$.

	CGH				CCG			
Instance #	LB	UB	Cpu (s)	Gap (%)	LB	UB	Cpu (s)	Gap (%)
7	359,274.6	359,348	3,361	0.02	359,348	359,348	649	0
8	306,950.3	307,070	7,931	0.04	307,058	**307,058**	4,701	0
9	332,919	332,952	4,006	0.01	332,952	332,952	236	0
34	281,604.3	281,688	3,448	0.03	281,668	**281,668**	674	0
35	278,818	278,818	1,129	0	278,818	278,818	519	0
36	264,507.2	**264,568**	2,990	0.02	264,158	264,650	14,400	0.15
61	290,602	290,648	10,064	0.02	290,626	**290,626**	4,285	0
62	336,986.7	336,990	1,434	0	336,990	336,990	252	0
63	290,772.4	290,808	8,205	0.01	290,808	290,808	1,549	0
88	502,313.3	504,552	14,408	0.45	504,334	**504,334**	595	0
89	445,792.6	448,960	14,404	0.71	448,880	**448,880**	499	0
90	510,238.7	531,584	14,423	4.18	531,566	**531,566**	1,644	0
115	437,182.5	437,212	2,515	0.01	437,212	437,212	172	0
116	482,243	482,316	2373	0.02	482,316	482,316	72	0
117	425,972	426,226	4,596	0.06	426,174	**426,174**	354	0
142	446,032	446,276	14,405	0.05	446,276	446,276	385	0
143	443,084	443,084	6,089	0	443,084	443,084	203	0
144	449,560.2	470,294	14,434	4.61	470,066	**470,066**	9,580	0
169	682,021.6	693,786	15,003	1.72	692,652	**693,296**	15,507	0.01
170	467,363.1	582,794	14,462	24.7	578,576	**578,576**	4,171	0
196	550,201.5	553,320	14,406	0.57	553,256	**553,256**	414	0
197	650,421	650,470	3,486	0.01	650,470	650,470	142	0
198	541,756	541,890	10,868	0.02	541,890	541,890	506	0
223	481,350.8	517,924	14,419	7.6	517,204	**517,204**	536	0
224	610,009.3	610,220	14,413	0.03	610,220	610,220	364	0
225	649,296	649,296	1,990	0	649,296	649,296	207	0

Large instances with trolley capacity $C = 30$ pose a greater computational challenge. For these cases, the optimal labeling scheme in our CCG algorithm cannot produce a lower bound within the 14,400 seconds CPU time limit. Consequently, we employ the CCG-H variant to assess whether it can yield acceptable upper bounds. Table 7.3 presents the upper bounds obtained by CCG-H, the CGH of Briant et al. [49], and the ILS of Van Gils et al. [48] for large instances with trolley capacity $C = 30$. CCG-H provides better UBs than the previous studies in 17 instances out of 27. All of these UBs are obtained in shorter CPU times than CGH of Briant et al. [49].

Table 7.3. Results on large instances with trolley capacity $C = 30$.

	CCG-H		ILS	CGH	
Instance #	UB	Cpu (s)	UB	UB	Cpu (s)
16	**304,834**	116	305,840	305,016	14,918
17	**267,318**	1,008	268,896	268,846	15,046
18	**255,400**	214	256,420	260,508	15,042
43	218,674	1,364	219,278	**218,018**	14,492
44	203,724	132	203,742	**203,030**	14,538
45	201,216	178	201,350	**200,998**	15,111
70	**231,054**	200	232,510	232,036	14,870
71	228,410	298	**228,188**	229,908	15,006
72	**238,402**	175	239,470	240,434	15,002
97	**340,732**	5,901	343,062	361,130	15,006
98	**454,510**	311	458,204	458,814	14,971
99	418,272	14,400	**417,336**	446,658	15,026
124	325,872	1,265	**322,434**	331,250	15,036
125	**320,568**	275	321,718	324,468	15,025
126	**346,246**	326	349,442	348,542	14,631
151	359,632	14,400	**358,618**	378,824	15,003
152	**358,732**	398	361,010	366,308	15,013
153	**378,306**	306	383,112	387,088	15,018
178	**474,508**	4,754	480,888	510,588	15,010
179	**524,338**	1,984	531,724	553,044	15,037
180	-	14,400	**600,422**	-	15,042
205	452,592	476	**451,856**	470,242	15,030
206	**501,434**	553	502,722	507,516	14,740
207	**498,200**	531	503,854	508,732	15,012
232	**522,534**	921	523,364	529,706	14,483
233	497,186	537	**497,086**	504,674	15,025
234	**447,772**	1,050	450,494	467,620	15,024

8. CONCLUSION

Order picking is one of the most labor-intensive processes in warehouses that relies on manual operations. As a result, optimizing this activity is essential for reducing operational costs and enhancing warehouse efficiency. The first part of this study addresses the order batching problem in conjunction with the selection of routing policies. Specifically, each batch of orders may follow a different routing strategy to minimize the overall distance traveled for item collection. We refer to this as the joint order batching and routing policy selection problem. To solve this problem, we propose two exact methods: one is based on column-and-cut generation and the other one is based on the branch-price-and-cut approach. Both are supported by a labeling algorithm that efficiently handles the pricing subproblem and the generation of valid cuts.

Our computational results not only identify the optimal routing policy for each batch under five distinct storage configurations, but also reveal that allowing flexible policy selection can reduce total travel distance by as much as 37.63% when compared to using a fixed routing policy, depending on the storage policy applied. The experimental design also enables direct comparisons across different storage strategies, with results highlighting the effectiveness of the within-aisle storage policy.

The routing strategies explored in this study—return, traversal, and largest gap—are straightforward yet effective pattern-based heuristics. As pointed out by Roodbergen and Koster [69], Petersen and Schmenner [23], and Glock et al. [70], pickers often find complex or optimal routing trajectories difficult to follow, which may lead to picking errors. A key contribution of this work is demonstrating that travel distance—and thus picking time—can be significantly reduced while maintaining the simplicity of picker routes by enabling the flexible use of these three routing heuristics.

In this thesis, we introduce the first exact solution method for the JOBBSPRP-D. The CCG algorithm successfully solves all small instances from Van Gils et al. [48] to

optimality within a maximum of 542 seconds. Among the 26 large instances with order size $|\mathcal{O}| = 100$ and batch capacity $C = 15$, 24 of these instances are solved optimally within the allowed CPU time. Notably, twenty-two of them are solved to optimality for the first time. Furthermore, our method provides for 12 instances tighter upper bounds than those given by any previous study, enabling a more accurate assessment of heuristic algorithms. For instances with order size $|\mathcal{O}| = 100$ and batch capacity $C = 30$, the CCG algorithm is unable to produce lower bounds within the time limit. To address this, we employ the heuristic extension CCG-H, which outperforms the methods of Van Gils et al. [48] and Briant et al. [49] by yielding better upper bounds in 16 out of 27 instances.

A promising direction for future research is to modify the objective function to focus on earliness and/or tardiness minimization. To our knowledge, no exact method has yet been developed for this extension. Although the deadlines used in the instances from Van Gils et al. [48] do not lead to infeasibility when treated as hard constraints, minimizing the earliness of order deliveries—alongside total picking time could significantly improve customer satisfaction. Both studies assume narrow aisles, allowing pickers to access both sides without additional movement. However, in practice, this can lead to conflicts when two pickers enter the same aisle simultaneously. Including this real-world constraint would increase problem complexity and necessitate more advanced solution techniques. Additionally, shifting from an offline to an online setting, where orders arrive dynamically, would align more closely with the real-world.

REFERENCES

1. Tompkins, J. A., J. A. White, Y. A. Bozer and J. M. A. Tanchoco, *Facilities Planning*, John Wiley & Sons, New Jersey, 2010.

2. Bartholdi, J. J. and S. T. Hackman, *Warehouse & Distribution Science*, Supply Chain and Logistics Institute, Atlanta, 2008.

3. Coyle, J. J., E. J. Bardi and C. J. Langley, *The Management of Business Logistics: A Supply Chain Perspective*, South-Western/Thomson Learning, Ohio, 2003.

4. Xie, L., H. Li and L. Luttmann, "Formulating and Solving Integrated Order Batching and Routing in Multi-Depot AGV-Assisted Mixed-Shelves Warehouses", *European Journal of Operational Research*, Vol. 307, No. 2, pp. 713–730, 2023.

5. Öncan, T., "MILP Formulations and an Iterated Local Search Algorithm with Tabu Thresholding for the Order Batching Problem", *European Journal of Operational Research*, Vol. 243, No. 1, pp. 142–155, 2015.

6. De Koster, R., T. Le-Duc and K. J. Roodbergen, "Design and Control of Warehouse Order Picking: A Literature Review", *European Journal of Operational Research*, Vol. 182, No. 2, pp. 481–501, 2007.

7. Scholz, A. and G. Wäscher, "Order Batching and Picker Routing in Manual Order Picking Systems: The Benefits of Integrated Routing", *Central European Journal of Operations Research*, Vol. 25, No. 2, pp. 491–520, 2017.

8. Kulak, O., Y. Sahin and M. E. Taner, "Joint Order Batching and Picker Routing in Single and Multiple-Cross-Aisle Warehouses Using Cluster-Based Tabu Search Algorithms", *Flexible Services and Manufacturing Journal*, Vol. 24, No. 1, pp. 52–80, 2012.

9. Oxenstierna, J., J. Malec and V. Krueger, "Efficient Order Batching Optimization Using Seed Heuristics and the Metropolis Algorithm", *SN Computer Science*, Vol. 4, No. 2, pp. 1–12, 2023.

10. Wagner, S. and L. Mönch, "A Variable Neighborhood Search Approach to Solve the Order Batching Problem with Heterogeneous Pick Devices", *European Journal of Operational Research*, Vol. 304, No. 2, pp. 461–475, 2023.

11. Gademann, N. and S. Velde, "Order Batching to Minimize Total Travel Time in a Parallel-Aisle Warehouse", *IIE transactions*, Vol. 37, No. 1, pp. 63–75, 2005.

12. Valle, C. A., J. E. Beasley and A. S. d. Cunha, "Modelling and Solving the Joint Order Batching and Picker Routing Problem in Inventories", *International Symposium on Combinatorial Optimization*, pp. 81–97, Salerno, Italy, 2016.

13. Valle, C. A., J. E. Beasley and A. S. Da Cunha, "Optimally Solving the Joint Order Batching and Picker Routing Problem", *European Journal of Operational Research*, Vol. 262, No. 3, pp. 817–834, 2017.

14. Briant, O., H. Cambazard, D. Cattaruzza, N. Catusse, A.-L. Ladier and M. Ogier, "An Efficient and General Approach for the Joint Order Batching and Picker Routing Problem", *European Journal of Operational Research*, Vol. 285, No. 2, pp. 497–512, 2020.

15. Jamal, J., D. Loske, M. Klumpp, X. Chou, A. Di Florio Di Renzo, M. Dell'Amico and R. Montemanni, "Skill-Based Joint Order Batching and Picker Routing Problem", *2022 The 9th International Conference on Industrial Engineering and Applications (Europe)*, pp. 64–69, Barcelona, Spain, 2022.

16. Heßler, K. and S. Irnich, *Modeling and Exact Solution of Picker Routing and Order Batching Problems*, Tech. rep., Technical Report LM-2022–03, Chair of Logistics Management, Gutenberg School, Tech. Rep., 2022.

17. Chabot, T., 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*, Vol. 69, No. 8, pp. 1242–1253, 2018.

18. Bahçeci, U. and T. Öncan, "An Evaluation of Several Combinations of Routing and Storage Location Assignment Policies for the Order Batching Problem", *International Journal of Production Research*, Vol. 60, No. 19, pp. 1–20, 2021.

19. Yang, N., "Evaluation of the Joint Impact of the Storage Assignment and Order Batching in Mobile-Pod Warehouse Systems", *Mathematical Problems in Engineering*, Vol. 2022, No. 1, p. 9148001, 2022.

20. Kübler, P., C. H. Glock and T. Bauernhansl, "A New Iterative Method for Solving the Joint Dynamic Storage Location Assignment, Order Batching and Picker Routing Problem in Manual Picker-to-Parts Warehouses", *Computers & Industrial Engineering*, Vol. 147, p. 106645, 2020.

21. Yang, M.-F., P.-H. Shih, J. C.-H. Pan and M.-C. Li, "The Optimal Layout Design for Minimizing Operating Costs in a Picker-to-Part Warehousing System", *The International Journal of Advanced Manufacturing Technology*, Vol. 118, No. 7, pp. 2523–2537, 2022.

22. Caron, F., G. Marchet and A. Perego, "Routing Policies and COI-Based Storage Policies in Picker-to-Part Systems", *International Journal of Production Research*, Vol. 36, No. 3, pp. 713–732, 1998.

23. Petersen, C. G. and R. W. Schmenner, "An Evaluation of Routing and Volume-Based Storage Policies in an Order Picking Operation", *Decision Sciences*, Vol. 30, No. 2, pp. 481–501, 1999.

24. Petersen, C. G. and G. Aase, "A Comparison of Picking, Storage, and Routing Policies in Manual Order Picking", *International Journal of Production Economics*,

Vol. 92, No. 1, pp. 11–19, 2004.

25. Dekker, R., M. De Koster, K. J. Roodbergen and H. Van Kalleveen, "Improving Order-Picking Response Time at Ankor's Warehouse", *Interfaces*, Vol. 34, No. 4, pp. 303–313, 2004.

26. Hsieh, L.-f. and L. Tsai, "The Optimum Design of a Warehouse System on Order Picking Efficiency", *The International Journal of Advanced Manufacturing Technology*, Vol. 28, pp. 626–637, 2006.

27. Dukic, G. and C. Oluic, "Order-Picking Methods: Improving Order-Picking Efficiency", *International Journal of Logistics Systems and Management*, Vol. 3, No. 4, pp. 451–460, 2007.

28. Chen, C.-M., Y. Gong, R. B. De Koster and J. A. Van Nunen, "A Flexible Evaluative Framework for Order Picking Systems", *Production and Operations Management*, Vol. 19, No. 1, pp. 70–82, 2010.

29. Chan, F. T. 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*, Vol. 38, No. 3, pp. 2686–2700, 2011.

30. Hsieh, L.-F. and Y.-C. Huang, "New Batch Construction Heuristics to Optimise the Performance of Order Picking Systems", *International Journal of Production Economics*, Vol. 131, No. 2, pp. 618–630, 2011.

31. Shqair, M., S. Altarazi and S. Al-Shihabi, "A Statistical Study Employing Agent-Based Modeling to Estimate the Effects of Different Warehouse Parameters on the Distance Traveled in Warehouses", *Simulation Modelling Practice and Theory*, Vol. 49, pp. 122–135, 2014.

32. Franzke, T., E. H. Grosse, C. H. Glock and R. Elbert, "An Investigation of the

Effects of Storage Assignment and Picker Routing on the Occurrence of Picker Blocking in Manual Ppicker-to-Parts Warehouses", *The International Journal of Logistics Management*, Vol. 28, No. 3, pp. 841–863, 2017.

33. Van Gils, T., K. Ramaekers, A. Caris and R. De Koster, "Designing Efficient Order Picking Systems by Combining Planning Problems: State-of-the-Art Classification and Review", *European Journal of Operational Research*, Vol. 267, pp. 1–15, 2018.

34. Haouassi, M., Y. Kergosien, J. E. Mendoza and L.-M. Rousseau, "The Integrated Orderline Batching, Batch Scheduling, and Picker Routing Problem with Multiple Pickers: the Benefits of Splitting Customer Orders", *Flexible Services and Manufacturing Journal*, Vol. 34, No. 3, pp. 614–645, 2022.

35. Muter, İ. and T. Öncan, "An Exact Solution Approach for the Order Batching Problem", *IIE Transactions*, Vol. 47, No. 7, pp. 728–738, 2015.

36. Muter, İ. and T. Öncan, "Order Batching and Picker Scheduling in Warehouse Order Picking", *IISE Transactions*, Vol. 54, No. 5, pp. 435–447, 2022.

37. Wahlen, J. and T. Gschwind, "Branch-Price-and-Cut-Based Solution of Order Batching Problems", *Transportation Science*, Vol. 57, No. 3, pp. 756–777, 2023.

38. Elsayed, E. and M.-K. Lee, "Order Processing in Automated Storage/Retrieval Systems with Due Dates", *IIE Transactions*, Vol. 28, No. 7, pp. 567–577, 1996.

39. Chen, T.-L., C.-Y. Cheng, Y.-Y. Chen and L.-K. Chan, "An Efficient Hybrid Algorithm for Integrated Order Batching, Sequencing and Routing Problem", *International Journal of Production Economics*, Vol. 159, pp. 158–167, 2015.

40. Žulj, I., H. Salewski, D. Goeke and M. Schneider, "Order Batching and Batch Sequencing in an AMR-Assisted Picker-to-Parts System", *European Journal of Operational Research*, Vol. 298, No. 1, pp. 182–201, 2022.

41. Henn, S., "Algorithms for Online Order Batching in an Order Picking Warehouse", *Computers & Operations Research*, Vol. 39, No. 11, pp. 2549–2563, 2012.

42. Ardjmand, E., H. Shakeri, M. Singh and O. S. Bajgiran, "Minimizing Order Picking Makespan with Multiple Pickers in a Wave Picking Warehouse", *International Journal of Production Economics*, Vol. 206, pp. 169–183, 2018.

43. Ardjmand, E., I. Ghalehkhondabi, W. A. Young II, A. Sadeghi, G. R. Weckman and H. Shakeri, "A Hybrid Artificial Neural Network, Genetic Algorithm and Column Generation Heuristic for Minimizing Makespan in Manual Order Picking Operations", *Expert Systems with Applications*, Vol. 159, p. 113566, 2020.

44. Saylam, S., M. Çelik and H. Süral, "The Min–Max Order Picking Problem in Synchronised Dynamic Zone-Picking Systems", *International Journal of Production Research*, Vol. 61, No. 7, pp. 2086–2104, 2023.

45. Cano, J. A., A. A. Correa-Espinal and R. A. Gómez-Montoya, "Mathematical Programming Modeling for Joint Order Batching, Sequencing and Picker Routing Problems in Manual Order Picking Systems", *Journal of King Saud University-Engineering Sciences*, Vol. 32, No. 3, pp. 219–228, 2020.

46. Rijal, A., M. Bijvank, A. Goel and R. De Koster, "Workforce Scheduling with Order-Picking Assignments in Distribution Facilities", *Transportation Science*, Vol. 55, No. 3, pp. 725–746, 2021.

47. Vanheusden, S., T. Van Gils, K. Braekers, K. Ramaekers and A. Caris, "Analysing the Effectiveness of Workload Balancing Measures in Order Picking Operations", *International Journal of Production Research*, Vol. 60, No. 7, pp. 2126–2150, 2022.

48. Van Gils, T., 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*, Vol. 277,

No. 3, pp. 814–830, 2019.

49. Briant, O., H. Cambazard, D. Cattaruzza, N. Catusse, A.-L. Ladier and M. Ogier, "A Bin-Packing Formulation for the Joint Order Batching, Picker Routing and Picker Sequencing Problem", *EURO 2022-32nd Conference of the Association of European Operational Research Societies*, Espoo, Finland, 2022.

50. Korbacher, L., K. Heßler and S. Irnich, *The Single Picker Routing Problem with Scattered Storage: Modeling and Evaluation of Routing and Storage Policies*, Tech. rep., Johannes Gutenberg University Mainz Gutenberg School of Management and Economics, Germany, 2023.

51. Ratliff, H. D. and A. S. Rosenthal, "Order-Picking in a Rectangular Warehouse: A Solvable Case of the Traveling Salesman Problem", *Operations Research*, Vol. 31, No. 3, pp. 507–521, 1983.

52. Baldacci, R., L. Bodin and A. Mingozzi, "The Multiple Disposal Facilities and Multiple Inventory Locations Rollon–Rolloff Vehicle Routing Problem", *Computers & Operations Research*, Vol. 33, No. 9, pp. 2667–2702, 2006.

53. Nemhauser, G. L. and L. A. Wolsey, *Integer and Combinatorial Optimization*, John Wiley & Sons, New York, 1988.

54. Baldacci, R., N. Christofides and A. Mingozzi, "An Exact Algorithm for the Vehicle Routing Problem Based on the Set Partitioning Formulation with Additional Cuts", *Mathematical Programming*, Vol. 115, No. 2, pp. 351–385, 2008.

55. Baldacci, R., A. Mingozzi and R. Roberti, "New Route Relaxation and Pricing Strategies for the Vehicle Routing Problem", *Operations Research*, Vol. 59, No. 5, pp. 1269–1283, 2011.

56. Glize, E., N. Jozefowiez and S. U. Ngueveu, "An ε-Constraint Column Generation-and-Enumeration Algorithm for Bi-Objective Vehicle Routing Problems", *Comput-*

ers & Operations Research, Vol. 138, p. 105570, 2022.

57. Muter, İ., "Exact Algorithms to Minimize Makespan on Single and Parallel Batch Processing Machines", *European Journal of Operational Research*, Vol. 285, No. 2, pp. 470–483, 2020.

58. Subramanyam, A., T. Cokyasar, J. Larson and M. Stinson, "Joint Routing of Conventional and Range-Extended Electric Vehicles in a Large Metropolitan Network", *Transportation Research Part C: Emerging Technologies*, Vol. 144, p. 103830, 2022.

59. Pessoa, A., R. Sadykov, E. Uchoa and F. Vanderbeck, "A Generic Exact Solver for Vehicle Routing and Related Problems", *Mathematical Programming*, Vol. 183, No. 1, pp. 483–523, 2020.

60. Queiroga, E., Y. Frota, R. Sadykov, A. Subramanian, E. Uchoa and T. Vidal, "On the Exact Solution of Vehicle Routing Problems with Backhauls", *European Journal of Operational Research*, Vol. 287, No. 1, pp. 76–89, 2020.

61. Damião, C. M., J. M. P. Silva and E. Uchoa, "A Branch-Cut-and-Price Algorithm for the Cumulative Capacitated Vehicle Routing Problem", *4OR*, Vol. 21, No. 1, pp. 1–25, 2021.

62. Contardo, C. and R. Martinelli, "A New Exact Algorithm for the Multi-Depot Vehicle Routing Problem Under Capacity and Route Length Constraints", *Discrete Optimization*, Vol. 12, pp. 129–146, 2014.

63. Barnhart, C., E. L. Johnson, G. L. Nemhauser, M. W. Savelsbergh and P. H. Vance, "Branch-and-Price: Column Generation for Solving Huge Integer Programs", *Operations Research*, Vol. 46, No. 3, pp. 316–329, 1998.

64. Achterberg, T., *Constraint Integer Programming*, Ph.D. Thesis, Technische Universit¨at Berlin, Fakult at II – Mathematik und Naturwissenschaften, Berlin, Germany, 2007.

65. Jepsen, M., B. Petersen, S. Spoorendonk and D. Pisinger, "Subset-Row Inequalities Applied to the Vehicle-Routing Problem with Time Windows", *Operations Research*, Vol. 56, No. 2, pp. 497–511, 2008.

66. Dolan, E. D. and J. J. Moré, "Benchmarking Optimization Software with Performance Profiles", *Mathematical Programming*, Vol. 91, pp. 201–213, 2002.

67. Pansart, L., N. Catusse and H. Cambazard, "Exact Algorithms for the Order Picking Problem", *Computers & Operations Research*, Vol. 100, pp. 117–127, 2018.

68. Cambazard, H. and N. Catusse, "Fixed-Parameter Algorithms for Rectilinear Steiner Tree and Rectilinear Traveling Salesman Problem in the Plane", *European Journal of Operational Research*, Vol. 270, No. 2, pp. 419–429, 2018.

69. Roodbergen, K. J. and R. Koster, "Routing Methods for Warehouses with Multiple Cross Aisles", *International Journal of Production Research*, Vol. 39, No. 9, pp. 1865–1883, 2001.

70. Glock, C. H., E. H. Grosse, R. M. Elbert and T. Franzke, "Maverick Picking: The Impact of Modifications in Work Schedules on Manual Order Picking Processes", *International Journal of Production Research*, Vol. 55, No. 21, pp. 6344–6360, 2017.

APPENDIX A: FIGURE PERMISSION

Within the scope of this thesis, the visuals produced and whose copyright has been transferred to the publisher have been used in the thesis book in accordance with the publisher's "publication policy on the reuse of texts and graphics produced by the author" stated on its website.