School of Electrical Engineering

Optimizing Warehouse Order Batching when Routing Is Precedence Constrained and Pickers Have Varying Skills

Marek Matusiak

Aalto University

DOCTORAL DISSERTATIONS

Aalto University publication series
DOCTORAL DISSERTATIONS 62/2014

Optimizing Warehouse Order Batching when Routing Is Precedence Constrained and Pickers Have Varying Skills

Marek Matusiak

A doctoral dissertation completed for the degree of Doctor of Science (Technology) to be defended, with the permission of the Aalto University School of Electrical Engineering, at a public examination held at the lecture hall AS1 of the school on the 6th of June 2014 at 12.

Aalto University
School of Electrical Engineering
Department of Electrical Engineering and Automation
Generic Intelligent Machines

Aalto University publication series
DOCTORAL DISSERTATIONS 62/2014

© Marek Matusiak

ISBN 978-952-60-5674-6
ISBN 978-952-60-5675-3 (pdf)
ISSN-L 1799-4934
ISSN 1799-4934 (printed)
ISSN 1799-4942 (pdf)
http://urn.fi/URN:ISBN:978-952-60-5675-3

Unigrafia Oy
Helsinki 2014

Finland

NORDIC ECOLABEL
441 697
Printed matter

Aalto University

Abstract

Aalto University, P.O. Box 11000, FI-00076 Aalto www.aalto.fi

Author
Marek Matusiak

Name of the doctoral dissertation
Optimizing Warehouse Order Batching when Routing Is Precedence Constrained and Pickers Have Varying Skills

Publisher School of Electrical Engineering

Unit Department of Electrical Engineering and Automation

Series Aalto University publication series DOCTORAL DISSERTATIONS 62/2014

Field of research Automation Technology

Manuscript submitted 28 April 2014 **Date of the defence** 6 June 2014

Permission to publish granted (date) 24 April 2014 **Language** English

[✓ **Monograph**] ☐ **Article dissertation (summary + original articles)**

Abstract

Warehouses are an important part of most supply chains. By batching customer orders and routing pickers effectively, warehouses aim to increase the efficiency of the order picking process. The contributions of this thesis are related to two extensions to the optimizing of picker-to-parts order picking operations in warehouses.

First, precedence constraints are introduced to picker routing, which pose new challenges to order batching algorithms. This is due to the relative complexity of precedence-constrained routing when compared to standard methods for routing pickers. The complexity is mitigated by the use of an effective savings estimate in calculating the properties of large batches, which reduces computation time significantly. This savings estimate is used in a Large Neighborhood Search algorithm, which outperforms heuristics from literature and compares well to optimal solutions (1.2% mean error). An A* algorithm is used for precedence-constrained picker routing. Compared to the performance of a reference warehouse, almost 16% (5000km) in total travel distance can be saved during a three month period.

Second, forecasting models of order pickers' batch execution times are built with multilevel modeling using a three-month set of operational data. It is shown that significant differences in picker performance exist. The forecasting models are used in an Adaptive Large Neighborhood Search algorithm in finding the right picker for the job. Compared to a state-of-the-art batching algorithm and to the current practice of assigning work in most warehouses, 9% is saved in total order picking time. These results show that differences among order pickers should be taken into account when optimizing order picking operations in warehouses. This is the first work where forecasting models are used to predict the performance of individual order pickers, and where such models are exploited with a job assigning algorithm.

Keywords warehousing, routing, order picking, metaheuristics, batching, multilevel modeling, worker modeling, business analytics, combinatorial optimization

ISBN (printed) 978-952-60-5674-6 **ISBN (pdf)** 978-952-60-5675-3

ISSN-L 1799-4934 **ISSN (printed)** 1799-4934 **ISSN (pdf)** 1799-4942

Location of publisher Helsinki **Location of printing** Helsinki **Year** 2014

Pages 137 **urn** http://urn.fi/URN:ISBN:978-952-60-5675-3

A? Aalto-yliopisto

Tiivistelmä

Aalto-yliopisto, PL 11000, 00076 Aalto www.aalto.fi

Tekijä	Marek Matusiak
Väitöskirjan nimi	Keräilyvaraston tilausten yhdistämisen optimointi keräilijöiden ominaisuudet huomioiden ja reitityksen ollessa etusijarajoitettua
Julkaisija	Sähkötekniikan korkeakoulu
Yksikkö	Sähkötekniikan ja automaation laitos
Sarja	Aalto University publication series DOCTORAL DISSERTATIONS 62/2014
Tutkimusala	Automaatiotekniikka
Käsikirjoituksen pvm 28.04.2014	**Väitöspäivä** 06.06.2014
Julkaisuluvan myöntämispäivä 24.04.2014	**Kieli** Englanti
☒ **Monografia**	☐ **Yhdistelmäväitöskirja (yhteenveto-osa + erillisartikkelit)**

Tiivistelmä

Varastot ovat tärkeä osa teollisuuden ja vähittäismyynnin toimitusketjuja. Varastojen keräilyprosessia on pyritty tehostamaan keruureittien optimoinnilla sekä yhdistämällä tilauksia isommiksi kokonaisuuksiksi, jotka voidaan kerätä yhden keruureitin varrelta lähtöpisteeseen välillä palaamatta. Tämän väitöskirjan tieteelliset kontribuutiot laajentavat tieteen nykyisiä varastojen vähittäiskeräilyprosessin optimoinnin käytäntöjä.

Kerättävien tuotteiden välillä voi olla etusijarajoitteita, ts. jotkut tuotteet tulisi kerätä ennen toisia. Perinteiset keräilijöiden reitittämiseen käytetyt algoritmit eivät ota tätä huomioon. Etusijarajoitettu reitittäminen on laskennallisesti verraittain raskasta. Tilausten yhdistämiseen käytetyt algoritmit kutsuvat reititysalgoritmia aina löytäessään uuden tilausyhdistelmän. Reitityksen ollessa etusijarajoitettua, jatkuva reitittäminen tekee kirjallisuudessa esitettyjen algoritmien käytöstä epäkäytännöllistä ratkaisuun kuluvan ajan vuoksi. Tässä työssä esitetään tapa vähentää laskennallisesti raskasta reititystä käyttäen säästettyyn matkaan perustuvia estimaatteja. Estimaatteja käytetään tilausten yhdistämisalgoritmissa varsinaisen reitityksen sijaan. Tämä nopeuttaa ratkaisun löytämistä laadusta tinkimättä: esitetty algoritmi löytää paremman ratkaisun tilausten yhdistämisongelmaan nopeammin kuin mikään testatuista verrokkialgoritmeista, varsinkin jos tilauksia on paljon. Optimaalisiin ratkaisuihin verrattuna, esitetty algoritmi pärjää hyvin, saavuttaen keskimäärin 1,2% epätarkemman tuloksen. Referenssivaraston alkuperäisten keruureittien kokonaismatkaan verrattuna säästöä kertyi 16%, eli noin 5000 km kolmen kuukauden ajanjaksona.

Kolmen kuukauden keruudatan pohjalta monitasomallinusmenetelmän avulla luodaan keruutehtävän kokonaisaikaa ennustavat kerääjäkohtaiset mallit. Ennuste riippuu kerääjästä ja työtehtävän parametreistä. Työntekijöiden välisten erojen tilastollinen merkittävyys osoitetaan varianssianalyysia käyttäen. Ennustemalleja käytetään adaptiivisessa hakualgoritmissa, joka etsii parhaita keräilijä/tilaus/tilausyhdistelmä-kombinaatioita isosta hakuavaruudesta. Verrattuna tieteen nykytilaan, jossa tilaukset yhdistetään kokonaismatkaa minimoiden, mutta ei oteta keräilijöiden taitoja huomioon, saavutetaan 9% säästöt kokonaiskeruuajassa. Nämä tulokset osoittavat, että keräilijöiden taidot tulisi ottaa huomioon keräilyvaraston toimintojen optimoinnissa.

Avainsanat varastointi, reititys, keräily, metaheuristiikka, tilausten yhdistäminen, monitasomallintaminen, työntekijöiden mallintaminen, liiketoimintatiedon hallinta, kombinatorinen optimointi

ISBN (painettu) 978-952-60-5674-6	**ISBN (pdf)** 978-952-60-5675-3	
ISSN-L 1799-4934	**ISSN (painettu)** 1799-4934	**ISSN (pdf)** 1799-4942
Julkaisupaikka Helsinki	**Painopaikka** Helsinki	**Vuosi** 2014
Sivumäärä 137	**urn** http://urn.fi/URN:ISBN:978-952-60-5675-3	

Preface

This research was conducted in the Centre of Excellence in Generic Intelligent Machines Research, partly funded by the Finnish Academy of Sciences, in the Department of Automation and Systems Technology during the years 2010-2014. Funding was in part received from the Motti project, which is a part of the Energy and Life Cycle Cost Efficient' Machines (EFFIMA) research program, managed by the Finnish Metals and Engineering Competence Cluster (FIMECC), and funded by the Finnish Funding Agency for Technology and Innovation (TEKES), research institutes and companies. Their support is gratefully acknowledged.

I would like to thank Professor Aarne Halme for the opportunity to pursue my Doctorate and for his role as my supervisor. Professor René de Koster of Erasmus University has been integral to the content of his thesis, you are a great mentor and an inspiration. My instructor Jari Saarinen inspired me to pursue this subject: thank you for telling me to leave the robots alone, and for the many ideas and discussions. The pre-examiners of this thesis, Professor Wout Dullaert and Assistant Professor Wilco van den Heuvel have my gratitude for taking the time to read through the thesis and helping to make it better.

Second, I would like to thank the people who helped with work done in this thesis. My father, Professor Jerzy Matusiak helped me to organize the thesis and supported me the whole way. My roommate at work, Eemeli Aro, with whom I had numerous discussions on my research; Janne Paanajärvi for being a fountain of knowledge in all things scientific; Professor Leo Kroon for modelling the precedence constrained routing and batching; my boss, Professor Ville Kyrki for his advice on things related to the thesis; Antti Maula for his help on many coding issues; Johan Grönholm for his work on building the distance matrix for the warehouse; Jani Mähönen of Rocla for presenting me with these warehousing problems (and for the data!); Olli Haavisto for his regression analysis primer; and Assistant Professor Murat Tarakci for the idea of using multilevel analysis to model the pickers.

The research done in this thesis would not have been possible without my periodic trips to the Erasmus University in Rotterdam to visit Professor René de Koster, funded by a generous and flexible travel grant from Teknillisen korkeakoulun tukisäätiö. Thank you.

Finally, I would like to thank my family: my wife Salla, who has been both amazing and amazingly supportive; my great kids for sleeping better of late; and my mother for always being there for me.

Espoo, May 7, 2014,

Marek Matusiak

Contents

List of Figures

List of Tables

List of Symbols

Symbol	**Description**
$\boldsymbol{a}_r$	Zero-one vector indicating which orders from $\mathcal{O}$ are included in batch r
$a_{o,r}$	Zero-one variable indicating whether order o is included in batch r
b	Bin (e.g., a roll container) to which a customer order is to be picked into
c_i	The cost to pick order i
$c_{i,j}$	The cost to pick a batch containing orders i and j
c_r	The cost to pick all items in batch r
$d_{g,k}$	Distance between items $g, k \in \mathcal{G}$
$d(m_i, \mathcal{V})$	Distance to state m_i for a set of orders $\mathcal{V}$
$\bar{d}(m_i, \mathcal{V})$	Estimated distance from state m_i to the goal for a set of orders $\mathcal{V}$
d_r	Travel distance for batch r
$e(s)$	Destroy function for a large neighborhood search algorithm for solution s
$f(s)$	Function which maps a solution s to its cost
g	An item (order line), contains information to which customer order it belongs to, the pick location and the number of units to be picked
$h(s)$	Neighborhood search heuristic (a *move*), which enables searching the neighborhood of s for new solutions
$i(s)$	Repair function for a large neighborhood search algorithm for partial solution s
l_r	Number of order lines (items) in batch r
$\boldsymbol{m}_r$	State vector of the A^*-algorithm in Section 3.2.2
o	Customer order

$o(g)$	Function indicating the customer order to which item g belongs to
p_g	Sequence number of item g
q_h	Probability of choosing heuristic h with the roulette wheel method
r	Batch (a combination of multiple customer orders)
$r(b)$	Function returning the batch to which bin b belongs to
s	Solution to a problem
s^*	Optimal solution to a problem
$t_{r,w}$	Batch execution time for picker w to complete batch r
$u_{0,w}$	Between group error term in a multilevel model for group (picker) w
$\boldsymbol{v}$	Vector collecting adjustable parameters for an Adaptive Large Neighborhood Search algorithm
v_e^r	Estimated batch rank for batch r
v_v^r	Real batch rank for batch r
w	An order picker
$\boldsymbol{x}_r$	Parameter vector for batch r
B	Distance matrix
C	Number of possible batch combinations in a solution (or part of it)
C_{n+}	Number of possible batch combinations in a solution for batches of size larger than n
D	Depot
$E(e^{\kappa_w})$	The Smearing Estimate (Duan, 1983) for picker w
I	Instance of a combinatorial optimization problem
J	Number of iterations of an algorithm
L_w	Productivity of picker w
M_{max}	Maximum working time during a virtual day
M_{min}	Minimum working time during a virtual day
M_w	Maximum working time for picker w
N	Maximum capacity of a picking device, i.e., the batch size
K_0	Starting temperature of a simulated annealing algorithm
P	Maximum number of items in a customer order
Q	Random number with a uniform distribution
R	Order picking wave size, i.e., the number of orders considered when running a batching algorithm

R^2	R-squared, the amount of variance explained by a regression model
R_c^2	Conditional r-squared, the amount of variance in a multilevel model explained by fixed and random effects
R_m^2	Marginal r-squared, the amount of variance in a multilevel model explained by fixed effects
$S_{i,j}$	Savings resulting from batching two customer orders i and j
S_{max}	Maximum savings
S_{new}	New savings
S_{org}	Original savings
S_{tot}	Total savings
$S_{\mathcal{P}}$	Savings resulting from batching the customer orders in set $\mathcal{P}$
S_r	Savings of batch r
SS	Sum of squared residuals
$\bar{S}_{i,j,k}$	Estimated savings resulting from batching customer orders i, j and k
$\bar{S}_{\mathcal{P}}$	Estimated savings resulting from batching the customer orders in set $\mathcal{P}$
T	Temperature in a simulated annealing routine
T_0	Initial temperature in a simulated annealing routine
T_n	Temperature in a simulated annealing routine at iteration no. n
V	Time complexity of an algorithm (big-oh)
U_g	Indicates the relative position of item g in the tour that picks g
X_r	Zero-one decision variable whether batch r is included in a solution
$X_{g,k,r}$	Zero-one decision variable whether batch r goes directly from item g to item k
$X_{w,r}$	Zero-one decision variable whether batch r assigned to picker w
$Y_{o,b}$	Zero-one decision variable whether order o is picked to bin b
$\mathcal{A}$	Set of all aisles
$\mathcal{B}$	Set of all bins
$\mathcal{G}$	Set of all items (order lines)
$\mathcal{G}_f$	Set of first items

$\mathcal{G}_l$	Set of last items
$\mathcal{G}_o$	Set of all items included in order o
$\mathcal{H}$	Set of all neighborhood search heuristics of an algorithm
$\mathcal{N}(s)$	The neighborhood of solution s
$\mathcal{N}_h(s)$	The neighborhood of solution s when searched with heuristic h
$\mathcal{L}_r$	Set of drop-off locations for batch r
$\mathcal{O}$	Set of all customer orders
$\mathcal{P}$	Set of batches
$\mathcal{R}$	Set of all possible batches
$\mathcal{R}^+$	Set of all batches with positive savings
$\mathcal{R}_e^+$	Set of all batches with positive estimated savings
$\mathcal{R}_v^+$	Set of all batches with positive real savings
$\mathcal{S}$	Set of feasible solutions for a problem
$\mathcal{V}$	Subset of customer orders $\mathcal{V} \in \mathcal{O}$
$\mathcal{W}$	Set of all order pickers
$\mathcal{Y}_r$	Set of all possible tours for batch r

Greek

$\beta_{i,w}$	Forecasting model coefficient i for picker w
δ	Number of iterations after which an update occurs
ϵ_r	Proportional batch rank error for batch r
γ_0	Common intercept for a multilevel model
κ_r	Within-group error term
λ	Decay parameter
μ	Mean value
ϕ	Cooling rate in geometric simulated annealing schedule
σ_1	Update to the score of a heuristic if a new solution is the global best
σ_2	Update to the score of a heuristic if a new solution is better than a current one
σ_3	Update to the score of a heuristic if a new solution is accepted
σ^2	Variance (within group)
τ_0	Between-group variation in intercepts
τ_1	Between-group variation in slopes
θ_h	The weight of a heuristic h
Ψ_h	Score of a heuristic

List of Acronyms

Acronym	Description
A*	An extension of Dijkstra's algorithm, which uses a cost-to-goal heuristic to minimize exploration
ABHC	Attribute Based Hillclimber
AGV	Automated Guided Vehicle
ANOVA	Analysis of Variance
ATSP	Asymmetric Traveling Salesman Problem
AIC	Akaike information criterion
ALNS	Adaptive Large Neighborhood Search
AS/RS	Automated storage and retrieval system
CPLEX	Optimization software by IBM
CV	Coefficient of variation, i.e., standard deviation divided by the mean.
C&W	Clarke and Wright savings algorithm
GAP	Generalized Assignment Problem
GLS	Generalized Least Squares
ICC	Intraclass Correlation Coefficient
LKH	Lin-Kernighan-Helsgaun algorithm, capable of solving TSP and ATSP problems
LNS	Large Neighborhood Search
OLS	Ordinary Least Squares
PCES	Precedence Constrained Estimated Savings based batching
REMIX	A random destroy-random repair neighborhood search heuristic
SA	Simulated Annealing

SA-REMIX	A Large Neighborhood Search algorithm, which uses the REMIX neighborhood search heuristic and a simulated annealing search scheme
SKU	Stock Keeping Unit
SOMB	Self-organization Map batching algorithm
SOP	Sequential Ordering Problem
TSP	Travelling Salesman Problem
VIF	Variance Inflation Factor
VLNS	Variable Large Scale Neighborhood Search
VNS	Variable Neighborhood Search
WMS	Warehouse Management System

Glossary

batch execution time the total time to complete a batch of orders, starting at the depot having empty roll containers on board the picking truck, and ending ending at the depot after the delivery of the orders.

batching the act of combining multiple customer orders that are to be picked simultaneously during one pick tour, with the aim of saving order picking time during a pick tour.

capability the ability of a worker to do a job (may be binary).

customer order consists of order lines, each line for a unique product or stock keeping unit (SKU), in a specified quantity (De Koster et al., 2007).

destroy a method for breaking apart some part of a solution.

distribution center an often large warehouse supplying one or multiple retailers.

fixed effect in multilevel modeling, the average model of all groups.

high-level a level of picking items that is reachable by pickers only with assistance (usually mechanical).

insertion see repair.

low-level a level of picking items that is reachable by pickers without assistance (usually mechanical).

order short for customer order.

parts-to-picker an order picking method, where the picker remains stationary at a pick station and items are transported to him/her via some, usually automated, method.

picker-to-parts an order picking method, where the picker travels between warehouse aisles to pick items.

precedence constraint a constraint specifying that a certain thing must be dealt with before some other thing.

random effect in multilevel modeling, the deviation from the fixed effect for a group.

removal see destroy.

repair a method for reinserting a destroyed part of a solution to form a new solution.

roulette wheel method a method of randomly choosing a neighborhood search heuristic from a set of heuristics based on the heuristic's performance scores.

skill the speed of at which a job can be carried out.

wave a set of orders that are processed together in an algorithm, see also wave picking.

wave picking the act of picking a certain number of orders for a common destination or a completion deadline, often started simultaneously by multiple pickers.

zone picking an order picking method where a picker is responsible for picks in a certain part, i.e., a "zone", in the warehouse.

1. Introduction

1.1 Background and Motivation

Supply chains are networks that are composed of various business entities (suppliers, manufacturers, distributors and retailers) who collaborate in an effort to: (i) acquire raw materials, (ii) convert the acquired materials to specified products, and (iii) deliver the products to retailers (Beamon, 1998). Traditionally, materials flow forward and information flows backward in the supply chain.

A supply chain is composed of two basic processes: the Production Planning and Inventory Control Process and the Distribution and Logistic Process, see Figure 1.1 (Beamon, 1998). Production planning describes the design and planning of the entire manufacturing process. Inventory control describes design and management of storage policies and procedures for raw materials and inventories of unfinished and finished products. The Distribution and Logistic Process specifies how products are transported from the warehouse to retailers. Products can be transported directly to retailers, or to a warehouse supplying the retailer (a *distribution center*).

Entities involved in a supply chain can use anticipatory or responsive business models, or a combination of both (Bowersox, 2011). Anticipatory business models are based on forecasting demand, producing or acquiring and storing the goods first and hoping that the forecasts hold. Responsive models sell first and manufacture and/or procure goods after the sale, delivering them to the customers as the goods become available. Most retailers use anticipatory business models, as they try to offer goods based on forecast demand (and sometimes the availability of goods).

Warehouses are an important part of most supply chains. More efficient

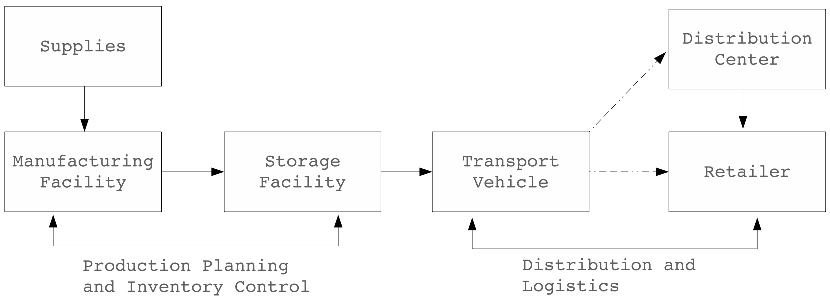


Figure 1.1. The Supply Chain Process. Adapted from Beamon (1998).

supply chains have resulted in smaller warehouses being replaced with fewer large warehouses. These distribution centers provide timely and economical inventory replenishment for retailers (De Koster et al., 2007, Bowersox, 2011). As warehouses become larger, companies are more reliant on their efficient functionality: in many cases, a company's whole supply chain is dependent on the performance of a single central warehouse. Between the time an order arrives at a warehouse and its eventual arrival at its destination, there is ample opportunity for errors and room for improvement (De Koster et al., 2007).

Warehouses can employ humans as pickers and/or be automated to some degree. When humans are employed, three picking methods can be distinguished: *picker-to-parts*, *parts-to-picker*, and *put systems* (De Koster et al., 2007). In picker-to-parts order picking, the picker drives or walks among the aisles to pick items. Parts-to-picker systems employ automated storage and retrieval systems (AS/RS) to bring the goods to the picker who works at the depot or a pick station. Finally, in put systems (order distribution systems), items for multiple orders are picked into a container either in a picker-to-parts or parts-to-picker manner, offered to an order picker for distribution over customer orders. *Zone picking* (Goetschalckx and Ashayeri, 1989) is a scheme where the picker is responsible for all picks in some area of the warehouse (such as a number of aisles) and places the picked goods on an automated guided vehicle (AGV) or a conveyor belt. Bartholdi et al. (2001) propose Bucket Brigade picking system, a type of work sharing zone picking system, where the pickers are not restricted to a single zone, but may operate in many, with the restriction that they may not pass each other. If the pickers are sequenced from slowest to fastest, a balance of work will spontaneously emerge resulting in high throughput.

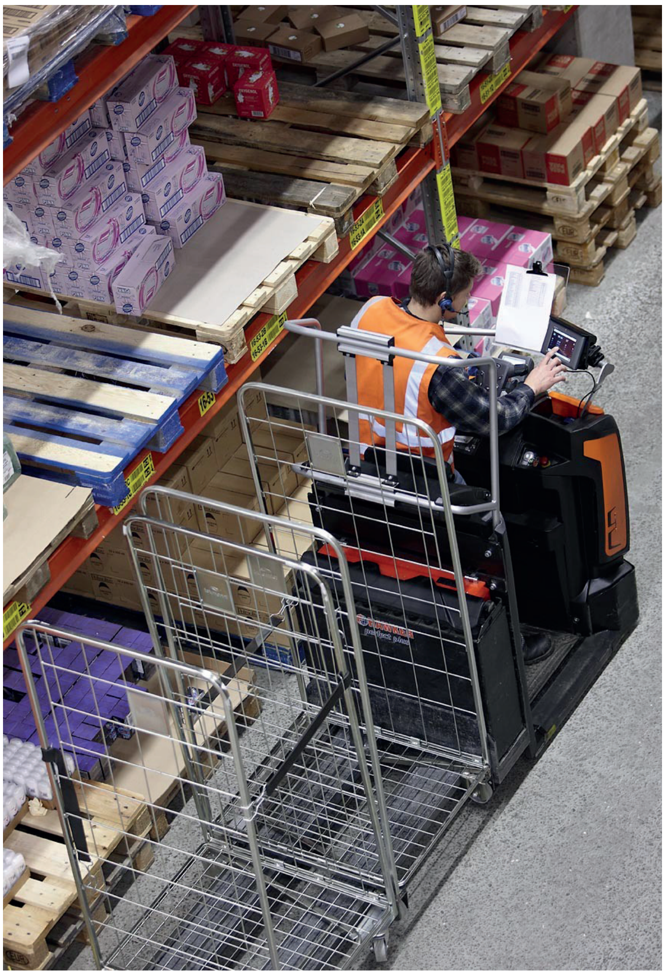


Figure 1.2. An order picker towing two empty roll containers. Image courtesy of Rocla.

This thesis focuses on optimizing processes in the type of warehouse most common in the world today: the *low-level* picker-to-parts warehouse, a type which, according to De Koster et al. (2007), forms a very large majority (approximately 80%) of all warehouses worldwide. The popularity of such warehouses can be attributed to the relative ease of setting up such a facility and the low initial capital cost when compared to AS/RS systems. In low-level picker-to-parts warehouses, the order picker picks requested items from storage racks or bins while traveling along the storage aisles (see Figure 1.2). Conversely, in *high-level* picker-to-parts warehouses, cranes or lifts mounted on order picking trucks are employed to lift the picker up so that he or she can reach all of the stored items (De Koster et al., 2007).

Tasks in a Warehouse There are many activities that are needed to get materials in and out of the warehouse. Listed roughly in the order of execution from the arrival to the departure of goods, and according to Tompkins et al. (2003), they are:

1. *Receiving* is a collection of the following activities: (*i*) the orderly receipt of all materials coming to the warehouse; (*ii*) assuring the quantity and quality of the items; and (*iii*) disbursing the materials.

2. *Inspection and quality control* are extensions of the receiving process and are done, e.g., when suppliers are inconsistent in quality or the goods must be inspected for some other reason.

3. *Repackaging* is done when goods are received in bulk and are to be stored in some other manner, e.g., in smaller units or with other products.

4. *Putaway* is the act placing materials in the warehouse.

5. *Storage* is the physical containment of the goods while it is awaiting demand.

6. *Order picking* is the process of removing items from storage to meet demand. It is the basic service around which most warehouse designs and processes are based.

7. *Postponement* is an optional step done after the picking process, where items are repackaged for more convenient use. Instead of doing this in the repackaging step, performing it after picking has the advantage of allowing more flexibility in the use of on-hand inventory.

8. *Sortation* can be done by the picker by picking items directly to the right customer order, or can be done after picking, if items are not picked by order, but by type, for example.

9. *Packing and shipping* can include the following tasks: (*i*) checking order completeness; (*ii*) packaging merchandise in containers appropriate for shipping; (*iii*) preparation of shipping documents; (*iv*) weighing

orders to determine shipping charges; (v) grouping orders to await an outbound carrier; and (vi) loading trucks.

10. *Cross docking* is the act of moving inbound goods directly to the shipping dock.

11. *Replenishing* involves moving items from reserve to primary picking locations.

Order Picking To remain competitive, warehouses need to continuously improve the efficiency of their processes. According to Gu et al. (2007), research on improving warehouse operation efficiency has focused on four areas: order picking, receiving, storage, and shipping. Of these four areas, order picking is the most important, and it is the most expensive process in distribution centers. De Koster et al. (2007) define order picking as "the process of retrieving products from storage (or buffer areas) in response to a specific customer request". In the literature, of the total cost of operating a distribution center, the cost of order picking is estimated to be as high as 55% (Drury, 1988) to 65% (Coyle et al., 1996). In spite of the advent of highly mechanized warehouses, orders are still picked manually in most warehouses, with pickers traveling in the warehouse to retrieve the items for an order (De Koster et al., 2007). These manual picking processes are increasingly supported by advanced technologies, such as pick-by-light systems, supporting mobile terminals, and voice-picking systems, which have made the order picking process more reliable and efficient (Berger and Ludwig, 2007, Weaver et al., 2010, Baumann, 2013). These computer-assisted picking processes generate extensive data logs, including who performed what pick at which location and at what point in time.

Travel time is the largest component of the total time of the order picking process (*batch execution time*), with a contribution of up to 50% (Tompkins et al., 2003). Much research has been devoted to methods reducing order pickers' travel times, including modifying the warehouse layout, using a different storage policy, batching orders, and routing pickers. For this thesis, the most relevant literature is related to the *batching* of *customer orders*. Customer orders consist of order lines, each line for a unique product or stock-keeping unit (SKU), in a specified quantity (De Koster et al., 2007). Order batching is defined as combining multiple customer orders into a single assignment for an order picker with the goal of minimizing travel or batch execution time. When optimizing for travel time, travel speed is generally assumed to be constant to justify travel distance based optimizations. Other components of total batch execution time, such as pick and setup time, are also usually assumed to be constants, and thus easily left out of optimization models. *Wave picking* is used if orders share a common destination or completion deadline. Customer orders belonging to a *wave* are processed, and often batched, together.

Customer orders can have *precedence constraints*, i.e., some items of the order must be picked to the relevant container before others. The problem of batching orders and routing pickers while respecting the precedence constraints of products is common, particularly in retail organizations, but such restrictions may also play a role in other warehouses (Dekker et al., 2004, Chan and Kumar, 2008). Precedence constraints may vary in nature. They may be due to weight restrictions (heavy products at the bottom of the roll container), fragility (light at the top), shape and size (big boxes at the bottom), stackability, but also preferred unloading sequence due to family grouping on the customer's shelves. Although order batching and picker routing have received attention in the literature, the combined problem of order batching and picker routing while respecting the precedence constraints of the products (and in this thesis, including potential multiple drop-off points in a route) has not. In the sample warehouse of this thesis, at most three customer orders (or *orders* for short) can be batched in a single pick tour, each possibly for a different customer (i.e., a store) and a different delivery location within the warehouse. Each customer order is to be preferably picked in a fixed sequence, due to family grouping in the company's stores.

Order pickers are people with different skill sets and varying physical capabilities. Some are experienced and others are new at their jobs. Interviews with warehouse managers and Warehouse Management System (WMS) suppliers reveal that in addition to the skills and physical capabilities of a picker, work motivation plays a key part in the picker's productivity. Besides batching and routing, the second key question of this thesis is how to make the most of the available workforce. Using regression analysis on detailed log data provides a way of modeling the pickers based on past performance as the data contain timestamps of all the pick events, travel distances, and the mass and volume of picked items. Using models forecasting the pickers' batch execution times based the on log data, the standard way of just minimizing travel time can be extended to optimizing the total order picking time. Furthermore, the models allow for better assignment of work among the pickers based on the strong areas of their past performances, and thus may motivate the pickers more.

1.2 Objectives

The general objectives of this thesis are to improve the efficiency of picker-to-parts order picking where the batching of customer orders plays an important part. In more detail, the objectives are:

- To include precedence-constrained routing in joint picker routing and order batching problems.
- Improve the efficiency of state-of-the-art order batching algorithms in the presence of precedence-constrained routing.
- To study order picker skill models based on operational data and to show that pickers have significant differences in skills, which affect expected job completion times based on the job parameters.
- To develop a methodology for order and batch assignment based on skills of individual pickers and to show that significant savings are achieved when compared to the current state-of-the-art.

1.3 Contributions of the Thesis

In this thesis, the following extensions to the related work presented in Section 1.4 are made.

This thesis introduces precedence constraints on picker routing. Precedence-constrained routing is combined with batching customer orders to find the combination of batches that maximizes the total travel distance savings of pick waves. As new batches are explored, their travel distance savings need to be calculated. This is usually done by routing the batches and comparing batched travel distance to the travel distance to pick individual orders in the batch. Items must be picked in a strict sequence to the customer orders. However, the complexity of precedence-constrained routing poses new challenges to order batching. Thorough exploration of combinatorial batch spaces with state-of-the-art batching algorithms becomes impractical, as these algorithms need to route each time a new batch is found. A new order batching algorithm is introduced to counter the time complexity of optimal precedence-constrained routing: a fast Large Neighborhood Search (LNS) algorithm, which uses an innovative way of estimating batch savings for batches of three or more customer orders to speed up the processing time. It is shown that this novel way of estimating batch savings is accurate for those batches that save the most in travel distance. Incidentally, the high-savings batches are the ones that generate the best solutions. For the precedence-constrained routing, an optimal A* algorithm is used.

In the presence of precedence constrained routing for medium to large wave sizes, the batching algorithm, Precedence Constrained Estimated Savings based batching (PCES), provides solutions of better quality than any tested state-of-the-art order batching algorithm. Only one of the tested algorithms, C&W(i) (Clarke and Wright, 1964, De Koster et al., 1999), is consistently faster than PCES, but it is much worse in solution quality in all cases. For small wave sizes, the mean error from optimal is 1.2%; and for medium and large wave sizes, PCES outperforms other methods by 1% to 3%. When compared to original batching and routing by the company, more than 15% in total travel distance is saved. It is also shown that the savings estimates and the batching part of PCES, SA-REMIX, can be applied to batching problems when the routing does not contain precedence constraints, with a similar solution quality as with the precedences.

Second, using actual picking data, multilevel modeling is used to build forecasting models for the pickers' batch execution times. It is shown that a significant amount of the total variance in batch execution time, 13%, results from differences in picker skill. To the author's knowledge, this is the first such application of using multilevel modeling to exploit operational data, at least in the warehousing context.

Third, the varying skills of the order pickers are exploited by extending the standard set-partitioning model (Gademann and Van de Velde, 2005) for the batching of orders with a generalized assignment problem. An Adaptive Large Neighborhood Search algorithm (ALNS) (Ropke and Pisinger, 2006a) is used to search a very large combinatorial space for solutions to assign orders and batches to the right pickers. Via the combination of the forecasting models and the ALNS algorithm, it is shown that almost 10% of total batch execution time can be saved by taking the skills of the pickers into account when compared to a state-of-the-art batching solution, but without considering picker skill. Compared to assigning work based on picker productivity (picking speed), taking skill into account saves 6% of the total time. To the author's knowledge, this is the first work where worker-specific forecasting models are used in conjunction with a work-assigning algorithm. Also, the proposed model and problem setup are novel. Finally, ALNS has not been previously applied to order picking problems.

1.4 Related Work

This section presents relevant related work. First, routing methods in picker-to-parts warehouses are presented, followed by routing with precedence constraints. Second, an overview of order batching methods is presented. Third, research where differences among workers skills and factors affecting workers' performances are discussed. Fourth, an overview of the class of Very Large Scale Neighborhood Search algorithms is presented. Multilevel modeling is briefly touched upon. Finally, a study where regression models of an order picking process have been built is discussed.

1.4.1 Routing in Picker-to-Parts Warehouses

A classic algorithm for order picking tour construction is the exact and polynomial time algorithm first presented in Ratliff and Rosenthal (1983) and extended in De Koster and Van der Poort (1998) to include multiple drop-offs. The algorithm presented in Ratliff and Rosenthal (1983) is further extended in Roodbergen and De Koster (2001b) to include a middle aisle. This extended version assumes a parallel-aisle warehouse with a maximum of three cross-aisles (see Figure 1.3) and does not account for precedence constraints or unidirectional travel.

Next to optimal routes, heuristics are often used. In S-shape routing (Randolph, 1993), also known as traversal routing, an order picker travels the whole subaisle if he or she needs to pick an item in the subaisle visited. An exception is made for the last aisle if the number of aisles is odd. With largest-gap routing (Randolph, 1993), each subaisle, except for the first

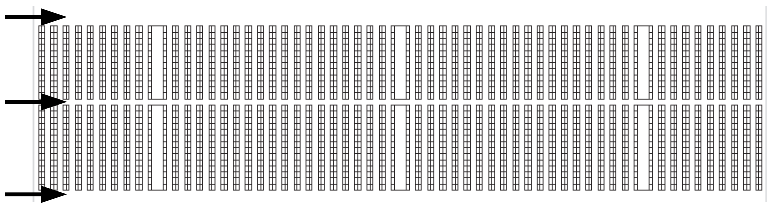


Figure 1.3. Top view of a warehouse three cross-aisles (marked with arrows).

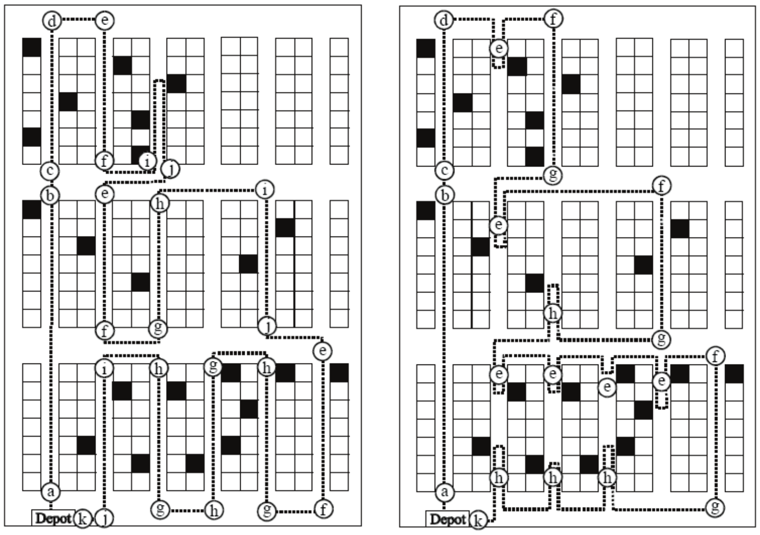


Figure 1.4. S-shape (left) and largest-gap (right) routing (Roodbergen and De Koster, 2001a).

and the last one visited, is exited on the entry side. Figure 1.4 shows examples of the S-shape and largest-gap routing methods. The first and the last visited aisles are traveled completely. A heuristic combining S-shape and largest-gap routing is presented in Roodbergen and De Koster (2001a). In aisle-by-aisle routing (Vaughan, 1999), each aisle is visited only once, i.e., first all items in the first pick aisle are picked, then all in the second, and so on. See Figure 1.5, with the optimal solution as a reference.

Theys et al. (2010) consider the applicability of the Lin-Kernighan-Helsgaun (LKH) TSP algorithm (Helsgaun, 2000) to routing order pickers. They compare LKH with several routing heuristics and obtain savings of up to 47% in travel distance. Helsgaun (2000) finds optimal solutions with the LKH algorithm for all previously solved TSP instances available at that time, including a 13,509-city problem, which was the largest problem instance solved to optimality at the time. LKH is used in Chapter 4 to test the SA-REMIX batching algorithm with non-precedence-constrained routing and as a reference algorithm in comparing the effectiveness of a

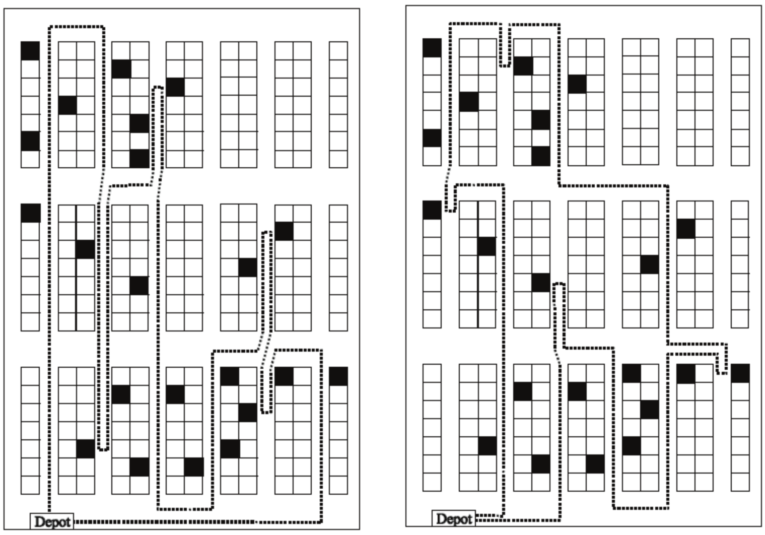


Figure 1.5. Aisle-by-aisle (left) and optimal (right) routing (Roodbergen and De Koster, 2001a).

routing heuristic used in Chapter 6.

1.4.2 Routing with Precedence Constraints

A dynamic programming algorithm for solving the single-vehicle many-to-many immediate request dial-a-ride problem (multiple customer destinations, each with a possibly unique drop-off location), which is similar to the A* routing algorithm presented in Section 3.2.2, is introduced in Psaraftis (1980b, 1983). In Psaraftis (1980b) each customer has three possible states: not picked up, picked up, and delivered. In Section 3.2.2, a single customer order is analogous to the customer, but with many more possible states than three. Furthermore, Psaraftis (1980b, 1983) does not use an A* type heuristic to speed up the search. The time complexity of Psaraftis' algorithm is $O(n^2 3^n)$ (where n is the total number of customers), and it solves problems for up to ten customers.

Kubo and Kasugai (1991) introduce the Precedence-Constrained Traveling Salesman Problem (PCTSP) and a branch-and-bound method for finding exact solutions for cases of up to 49 locations with acceptable computation times. The precedence-constrained path construction problem can be modeled as a case of the Sequential Ordering Problem (SOP) (Escudero, 1988). The SOP can be formulated as an Asymmetric Traveling Salesman Problem (ATSP) with precedence constraints. In the SOP, paths usually have a start and a finish position that differ from each other, while ATSP paths finish where they started. In the general ATSP case, each of the non-visited cities can be the next target with each iteration, but in the SOP, the set of the next possible cities is limited as defined by a directed graph formed from the problem. The routing problem presented in Section 3.2.2 is a special case of the one presented in Kubo and Kasugai (1991) since items in each customer order must be picked in a strict sequence.

1.4.3 Order Batching

The savings algorithm by Clarke and Wright (1964), C&W(i), as well as an extension of it, C&W(ii), are used for batching orders in De Koster et al. (1999). They are compared to *seed* algorithms using two routing strategies: S-shape and largest-gap. Seed algorithms consist of two distinct steps: seed order selection and order addition. A single order is selected as the seed order based on criteria, e.g., the largest number of items, the longest travel time, or the farthest item. Additions can be done using different rules such as adding the order that minimizes the sum of the distances of every item of the seed and the closest item in the order, or minimizing the additional number of aisles to be traveled. The authors find that seed algorithms work best with S-shape routing and large pick device capacity, while savings algorithms work best with largest-gap and small pick capacity. C&W(ii) consistently outperforms C&W(i), but is computationally more expensive.

Albareda-Sambola et al. (2009) use a Variable Neighborhood Search (VNS) algorithm to batch orders. It uses six different local exchange schemes incorporated into three different search neighborhoods of varying sizes to find good batches. Within each neighborhood, all moves belonging to that neighborhood are tried, and the one that results in the largest savings is chosen. The larger neighborhoods are searched as needed. If a current one fails to produce a better solution, the next (larger) neighborhood is explored for a better solution until no improvement can be made. Albareda-Sambola et al. (2009) compare VNS to the C&W(i), C&W(ii), and seed algorithms, and it consistently outperforms them. The authors find that the best performing algorithm from the literature is C&W(ii), which is on average 2% worse than VNS. Solution quality comes with added computational complexity. When compared to C&W(ii), a much larger part of the combinatorial batch space is explored. For the most complex instance run (250 orders), VNS took almost six times as much time to reach its solution. For routing batches, the authors use the computationally inexpensive S-shape, largest-gap, and combined heuristics.

Henn and Wäscher (2012) introduce an Attribute Based Hill Climber (ABHC) order batching algorithm (Whittley and Smith, 2004). This ABHC uses a set of attributes to guide the search out of local minima and a set of moves to explore the neighborhood of a current solution. Two moves are used to search for new solutions: shifting an order from one batch to another batch and swapping two orders between two batches. As the attribute that gave the best results, the authors used a pair of customer orders. Each attribute is given a value, which initially equals infinity. As a solution (a set of batches) is found, it will have a total travel distance. That solution is accepted if at least one of the attributes has a current value that is higher than the current solution's value (the total travel distance), after which that attribute's value is set to the current value. S-shape and largest-gap routing are used as routing algorithms. Comparisons are made between three local search algorithms, a Tabu Search heuristic, and C&W(ii). ABHC compares favourably to these algorithms, with over 2% better results than those achieved with C&W(ii).

A method for batching orders is introduced in Gademann and Van de Velde (2005). The order-batching problem is modeled as a set-partitioning problem. A column-generation algorithm is used to solve the linear programming relaxation. They rely on the polynomial time algorithm presented in Ratliff and Rosenthal (1983) to calculate the route length. They find that the maximum batch size has the largest impact on the solution time.

Large-scale order batching in parallel aisle warehouses is considered by Hong et al. (2012b). The authors introduce a route-selecting order batching (RSB) formulation, a route bin packing formulation for calculating lower bounds, and a heuristic route-packing based order batching procedure (RBP). Instead of constructing routes for batches, they use the S-shape routing method to construct a set of all possible routes given a warehouse layout to which batches can be assigned. Given a batch, a best-fit route can be selected with RSB. They assume that each order picker can pick up to ten orders to ten bins, and an average order size is two lines. A comparison of RBP is made to C&W(ii) and seed algorithms, with the RBP dominating the other algorithms and C&W(ii) performing better than the seed heuristics.

In comparison to the batching methods in the literature, which use multiple deterministic neighborhood search heuristics, the method presented in Chapter 3, PCES, uses a single flexible stochastic heuristic based on random moves, and is not constrained to a set of deterministic moves. Other similar methods (Gademann and Van de Velde, 2005, Albareda-Sambola et al., 2009, Henn and Wäscher, 2012) do explicit batch evaluation (and thus routing), which is computationally expensive, while PCES estimates the best batches using information inherent in batches of one and two customer orders. Furthermore, the problem in Chapter 3 also includes precedence constraints in the routing. With computationally more expensive routing algorithms, such as those used in Chapters 3 and 4, solving the batching with any such deterministic batching algorithm becomes intractable due to exponential growth in calls to routing algorithms. In combination, the flexible heuristic along with the estimate enables escaping local minima and exploring larger parts of the set of possible batches more thoroughly for good options. In addition to good solution quality, PCES is also computationally fast. With precedence-constrained routing, it is slower than C&W(i) but faster than C&W(ii) and provides good solutions to instances that C&W(ii) cannot solve in a reasonable time.

A simulated annealing (SA) algorithm for order batching is introduced by Hong et al. (2012a). The authors deal with narrow-aisle systems with the possibility of picker blocking. In addition to the batching problem, the authors also deal with the sequencing problem, which is also NP-hard. They use four neighborhood search heuristics in their SA-algorithm: the first one exchanges two orders from different batches; the second swaps the sequence of two batches; the third and fourth impose an acceptance condition on the order exchange to limit the size of neighborhoods to be searched. Their method aims to minimize total retrieval time, which is the sum of pick and travel time and time due to delays. The algorithm uses a geometric cooling schedule (Cohn and Fielding, 1999), while the criteria for accepting a worse batch are the same as PCES. Hong et al. (2012a) use computationally light traversal routing, and thus have no need to make use of savings estimates. Similarly, as in Chapter 6, the simulated annealing algorithm presented in Hong et al. (2012a) solves a combination of two NP-hard problems. They consider assignments to a set of homogeneous pickers, while the main contribution of Chapter 6 is to extend the order batching problem to include the (generalized) assignment of orders and batches to order pickers of varying skills.

Hsieh and Huang (2011) introduce two batch-construction heuristics based on data-clustering methods: K-means Batching based on the

K-means algorithm by MacQueen (1967); and Self-organisation Map (SOMB) based on the Self-Organizing Map (Kohonen, 1990). The K-means algorithm can be classified as a seed algorithm. It uses a selection rule of selecting the order with the minimum travel distance and an addition rule of adding an order which causes the minimum increment in travel distance when combined with the existing batch. The process is greedy and sequential and very similar to seed rules presented previously in De Koster et al. (1999). The SOMB algorithm groups orders by an order-relativity measure using the SOM. The measure is a weighted sum of the number stock-keeping units (SKUs) in an order and the number of the same aisles covered between two orders.

Of these algorithms, C&W(i) and C&W(ii), the VNS of Albareda-Sambola et al. (2009), and the ALNS by Henn and Wäscher (2012) are used in comparison algorithms to the batching algorithm in Chapter 3. C&W(i) is used in constructing initial batching solutions in chapters 3 and 6, and the VNS in Chapter 6. These algorithms are presented in more detail in Chapter 2.

1.4.4 Heterogeneous Workers and Jobs

Assigning jobs to workers based on their *capabilities* has been studied relatively well. A distinction is made between capability (the degree to which a person is able to carry out a certain task, regardless of the time it takes) and *skills* (the speed at which such a task can be carried out). In this thesis, all pickers have the capability to pick the batches, albeit at different speeds. Differences in skills may be caused by, the ability to lift heavy objects, motivation, stacking ability, and knowledge of the warehouse — many of which are not directly quantifiable.

Taking advantage of workers' capabilities has been studied and modeled as a generalized assignment problem (GAP). For extensive surveys on the subject, see Cattrysse and Van Wassenhove (1992) and Pentico (2007). Campbell and Diaby (2002) solve a GAP for cross-trained workers with an assignment heuristic. Each worker has a predetermined value from zero to one, characterizing his or her ability to work in a department (to do a job). Chapter 6 extends the order batching problem with the GAP to include knowledge of the pickers' skills in batching and assigning orders to pickers.

Another stream of research investigates how worker speed depends on endogenous and exogenous factors (Bendoly et al., 2006). Powell and

Schultz (2004) show that workers increase speed in the presence of a visible backlog. Doerr and Arreola-Risa (2000) investigate how the variability of task completion times is affected by different tasks and workers. They find that the most significant factors affecting the variability are: (1) the worker who did the work and (2) the interaction effect between the worker and the task, while the task type in question had no significant influence on its own. Juran and Schruben (2004) use personality and demographic data to predict task execution times of a collaborative two-worker task. They find that including information on individual workers has significant impact on simulation accuracy compared to assuming no differences among workers. Bartholdi and Eisenstein (1996) show that if workers are sequenced from slowest to fastest in typical production line, a stable partition of work will spontaneously emerge, with the production rate converging to the maximum possible value. These studies reinforce the hypothesis of picker skills playing a significant role in total batch execution time, which is further studied in chapters 5 and 6.

1.4.5 Very Large Scale Neighborhood Search

Very Large Scale Neighborhood Search (VLSN) is a class of neighborhood search algorithms, where the searchable neighborhood size grows exponentially with the instance size or the neighborhood is too large to be searched explicitly (Pisinger and Ropke, 2010). Ahuja et al. (1998) classify VLSN methods into three categories: variable-depth methods, network-flow based improvement methods, and methods based on subclasses solvable in polynomial time. Pisinger and Ropke (2010) extend these categories by including Large Neighborhood Search (LNS) algorithms.

The joint batching and generalized assignment problem dealt in Chapter 6 belongs to this class, as does the order batching problem when batch size grows, especially when batch evaluation is computationally expensive, which imposes a practical limit on explicitly searching the whole neighborhood (see Chapter 3). To solve the order batching part in Chapter 3, a LNS-type algorithm is applied. In Chapter 6, an Adaptive Large Neighborhood Search (ALNS) algorithm (an extension of LNS) is used to solve a joint batching and generalized assignment problem. The LNS and ALNS algorithms are detailed more thoroughly in sections 2.2.5 and 2.2.6, respectively.

Large Neighborhood Search

According to Pisinger and Ropke (2010), neighborhoods in LNS (Shaw, 1997) methods are defined by *destroy* and *repair* methods. They are also called in *removal* and *insertion* methods in other sources, e.g., in Ropke and Pisinger (2006a) and Gharehgozli et al. (2013). The destroy methods of neighborhood search heuristics break down a part of a solution in a predefined manner, while the repair methods try to find good alternatives to a solution by constructing another feasible solution in the neighborhood of the solution based on the repair method associated with the neighborhood search heuristic. These methods typically contain some element of randomness in choosing what parts of a solution to modify. The degree of destruction (Pisinger and Ropke, 2010) of the destroy method should be carefully chosen. Destroying too small parts of a solution can result in myopic exploration, while too large changes can result in suboptimality and repeated re-optimization.

Adaptive Large Neighborhood Search

Adaptive Large Neighborhood Search, first proposed in Ropke and Pisinger (2006a) to solve the Pickup and Delivery Problem with Time Windows, extends the LNS by adding a learning component to the search. Typically, ALNS uses several heuristics to form the search neighborhood. Details of each heuristic's runtime performance are kept in memory. Each heuristic has a score, which gets updated positively if good solutions are found and/or the heuristic finds new interesting solutions. One heuristic is selected with each iteration, with a weighted random selection criterion called the *roulette wheel method* (Pisinger and Ropke, 2010). As with the LNS, a heuristic is generally composed of a combination of destroy and repair methods. However, in Section 6.3.4, dedicated heuristics are used, which are not naturally divisible into both destroy- and repair-type methods. The set of all heuristics $\mathcal{H}$ is formed by (not necessarily all) combinations of the destroy and repair methods and the possible dedicated heuristics. ALNS has been shown to work very effectively on a wide variety of computationally hard problems, such as container scheduling (Gharehgozli et al., 2013), vehicle routing (Ropke and Pisinger, 2006a) and scheduling technicians and tasks with varying skills (Cordeau et al., 2010).

1.4.6 Multilevel Modeling

Multilevel modeling is a regression analysis technique, that can be used to distinguish group level effects in some population. A standard linear multivariate regression can be extended to a linear multilevel model by considering varying intercepts and slopes for each group. *Fixed effects* of a multilevel model contain the overall (mean) model encompassing information from all groups, quite similarly as a single multivariate model would. In linear multilevel regression, the *random effects* or variations to the fixed effect intercept and slope are calculated for each group. In this manner, forecasting models with group-specific parameters can be found.

1.4.7 Forecasting Order Picking Time

Larco Martinelli (2010) studies the problem of how to assign products to different storage locations, with the objective to minimize both order picking cycle time and worker discomfort. By reducing discomfort felt by the workers, the well-being of the workers increases, and may also yield the long-term benefits of increased productivity (Kuijt-Evers et al., 2007). Larco Martinelli (2010) uses extensive warehouse log data to build regression models for both objectives. Subsequently, the models are used in a dual-objective assignment model to optimize product placement. Larco Martinelli (2010) finds that there is an efficient frontier between the two objectives. If either cycle time is solely prioritized, worker discomfort is high and vice versa.

1.5 Outline

Chapter 2 presents algorithm descriptions and mathematical models related to the subject. Chapters 3 to 6 contain the scientific contributions of this thesis. Chapter 7 concludes the thesis.

The main contribution in Chapter 3 is the modeling and solving of a joint precedence-constrained picker routing and order batching problem. A journal article based on this chapter has been published: (Matusiak et al., 2014). These results are extended in Chapter 4 by relaxing the precedence constraints in the routing. In Chapter 5, the large order picking data-set obtained from the Finnish retailer is processed and used to form forecasting models for the execution times of batches for pickers. The forecasting models constructed in Chapter 5 are utilized in Chapter 6 to solve a joint-routing, order batching and generalized assignment problem.

1.6 Author's Contribution

Chapter 2 presents mathematical models and algorithms relevant to this thesis. All the models in that chapter are from the literature. The algorithm in Section 2.2.1 has been developed by the author. The other algorithms presented in Chapter 2 have been implemented by the author, as detailed in the relevant references. Chapters 3 to 7 hold the main contributions of the thesis. These chapters are the work of the author, apart from the exceptions noted below. The problem description and mathematical model presented in Section 3.1 and the CPLEX code used in a part of the experiments in Chapter 3.3 were formulated and coded by Professor Leo Kroon. The Lin-Kernighan-Helsgaun algorithm for solving the Traveling Salesman Problem, used in Chapter 4 and Chapter 6, has been developed by Keld Helsgaun and is free for research use.

1.7 Assumptions

There are number of assumptions made in this thesis. In Chapter 3, the pickers are always assumed to travel with a constant speed, the time to pick an item to the container is assumed constant, as is the setup time (getting on and off the truck). Congestion is not considered in the optimization, as are no other dynamic factors. Also, it is assumed that the order pickers are alike. The mathematical model does not consider time windows for the deliveries' of the orders, as the log data does not contain information on the truck departures.

In Chapter 5, the standard assumptions of non-multicollinearity, normality, and non-heteroscedasticity are made in the regression. However, the data are heteroscedastic, which is partly compensated by the logarithmic transformations applied to the data.

Some of the assumptions in Chapter 6 are the same as in 3: congestion, deadlines for orders and system dynamics are not considered. While batches are assigned, they are not sequenced in any manner. Stochasticity in the picking process is not taken into account, i.e., the problem is solved in a deterministic manner not considering the uncertainties that could be extracted from the regression model.

2. Relevant Mathematical Models and Algorithms

This chapter is organized as follows. In the first part of this chapter, combinatorial optimization problems are formally defined. The presented nomenclature is used in problem definitions later. Three separate combinatorial optimization problems are described to illustrate subproblems, each of which plays a part in the joint models presented in chapters 3, 4 and 6. These are the traveling salesman problem, the order batching problem, and the generalized assignment problem.

The second part of this chapter starts with descriptions and pseudocode of some of the algorithms implemented for comparison purposes and initial solutions. Finally, Large Neighborhood Search and Adaptive Large Neighborhood Search metaheuristics are detailed, which are later applied to order picking problems in chapters 3, 4 and 6.

2.1 Mathematical Models

2.1.1 Combinatorial Optimization Problems

Following Pisinger and Ropke (2010), let I be an instance of a combinatorial optimization problem, where $\mathcal{S}$ is the set of feasible solutions for the problem. Let $f : \mathcal{S} \rightarrow \mathbb{R}$ be a function which maps a solution to its cost. A *globally optimal* solution s^* is such that cost is minimized: $f(s^*) \leq f(s), \quad \forall s, s^* \in \mathcal{S}$. A *neighborhood* of a solution $s \in \mathcal{S}$ is defined as $\mathcal{N}(s) \in \mathcal{S}$. $\mathcal{N}$ is function which maps s to a set of solutions $\mathcal{S}' \subseteq \mathcal{S}$. If $f(s') \leq f(s), \forall s' \in \mathcal{N}(s)$, then s' is said to be a *locally optimal* solution for neighborhood $\mathcal{N}(s)$.

A *neighborhood search heuristic* $h(s) \in \mathcal{H}$, or a *move*, is a function which enables searching of the neighborhood $\mathcal{N}_h(s)$ of a solution $s \in \mathcal{S}$. $\mathcal{H}$ is the set of all neighborhood search heuristics of an algorithm.

A steepest descent algorithm is an algorithm explores $\mathcal{N}(s)$ for solution s. In this case, $\mathcal{N}(s)$ is searched for the best solution $s' \in \mathcal{N}(s)$, i.e., $s' = \arg\max_{s'' \in \mathcal{N}(s'')} f(s'')$. If $f(s') < f(s)$, an update $s = s'$ is performed.

2.1.2 Routing

The problem of routing order pickers in conventional multi-parallel-aisle warehouses is often modeled as a Steiner Traveling Salesman Problem (Steiner-TSP) (Cornuéjols et al., 1985, De Koster et al., 2007, Theys et al., 2010). The Steiner-TSP includes Steiner and non-Steiner nodes: *Steiner nodes* can be visited multiple times, and are typically located at the end of aisles. Each *non-Steiner node* represents a Stock Keeping Unit's (SKU) location from which items need to be picked during the current tour. Theys et al. (2010) propose that for warehouses with unconventional layouts, the Steiner nodes can be discarded. Let $\mathcal{G}'$ be the set of items to picked during a tour. Among other things, an item contains information about location of the products to be picked. Let D denote the depot in the warehouse, and let $\mathcal{G} = \mathcal{G}' \cup D$. Also, let $X_{i,j}$ be a binary decision variable, which equals one if item i is picked directly before item j, and zero otherwise. The problem can now be represented as a standard TSP:

$$\min \sum_{i \in \mathcal{G}} \sum_{j \in \mathcal{G} \setminus i} c_{i,j} X_{i,j}, \tag{2.1}$$

subject to

$$\sum_{i \in \mathcal{G}} X_{i,j} = 1 \quad \forall j \in \mathcal{G} \setminus i, \tag{2.2}$$

$$\sum_{j \in \mathcal{G}} X_{i,j} = 1 \quad \forall i \in \mathcal{G} \setminus j, \tag{2.3}$$

$$U_1 = 1, \tag{2.4}$$

$$2 \leq U_i \leq |\mathcal{G}| \quad \forall i \neq 1, \tag{2.5}$$

$$U_i - U_j + 1 \leq (\mathcal{G} - 1)(1 - X_{i,j}), \tag{2.6}$$

$$X_{i,j} \in \{0, 1\} \quad \forall i, j \in \mathcal{G}, \tag{2.7}$$

where (2.2) and (2.3) guarantee that each item location is entered and exited once, respectively. Equations (2.4), (2.5), and (2.6) are the subtour elimination constraints of Miller et al. (1960). If for some $i, j \in \mathcal{G}$, $c_{i,j} \neq c_{j,i}$, the problem is said to be asymmetric; otherwise it is symmetric.

2.1.3 Order Batching

A customer order $o \in \mathcal{O}$ consists of a subset $\mathcal{G}_o$ of the set of all *items* (*order lines*) $\mathcal{G}$ to be picked during some period of time, where $\mathcal{O}$ is the set of all customer orders. The items in each order have to be picked to the same container, or in the case of the sample retailer, to a roll container. A combination of customer orders that can be picked together during a pick tour by one order picker is a *batch*, and the set of all possible batches is denoted by $\mathcal{R}$. For each feasible batch $r \in \mathcal{R}$, a version of a Traveling Salesman Problem (TSP, see Section 2.1.2) can be defined. These TSPs are usually solved with heuristics (Ratliff and Rosenthal, 1983, Randolph, 1993, Theys et al., 2010), or in some special cases with optimal algorithms (Ratliff and Rosenthal, 1983, De Koster and Van der Poort, 1998, Roodbergen and De Koster, 2001b). In the commonly used order batching model of Gademann and Van de Velde (2005), presented below, solving the TSP is included implicitly only as the travel cost. This is not exactly correct, but it does make the model much simpler when the focus is not so much on the routing, but on the order batching algorithm. In Chapter 3, a joint precedence-constrained TSP and order batching problem is presented, while in the model of Chapter 6 the TSP is similarly absent from the model as below.

Following Gademann and Van de Velde (2005), the *order batching problem*, which is central for the research presented in this thesis, can be modelled as a set partitioning problem. Every batch $r \in \mathcal{R}$ is characterized by a zero-one vector $\boldsymbol{a}_r = (a_{1,r}, ..., a_{|\mathcal{O}|,r})$, where $a_{o,r} = 1$ if and only if order $o \in \mathcal{O}$ is included in batch r. Batch r is feasible only if $\sum_{o \in \mathcal{O}} a_{o,r} \leq N$, where N is the maximum customer order capacity of the picking device (the *maximum batch size*). Let c_r be the cost to pick all items of batch r. Furthermore, let X_r be a zero-one variable, such that

$$X_r = \begin{cases} 1 & \text{if batch } r \text{ is selected} \\ 0 & \text{otherwise.} \end{cases} \tag{2.8}$$

In order to minimize the total cost to pick all batches, the model can be described as follows:

$$\min \sum_{r \in \mathcal{R}} c_r X_r, \tag{2.9}$$

subject to

$$\sum_{r \in \mathcal{R}} a_{o,r} X_r \quad \forall o \in \mathcal{O}, \tag{2.10}$$

$$X_r \in \{0,1\} \quad \forall r \in \mathcal{R}, \tag{2.11}$$

where (2.10) guarantees that each order is assigned to one batch and one batch only. Gademann and Van de Velde (2005) show that for batch sizes larger than two ($N > 2$), the order batching problem is NP-hard.

2.1.4 Generalized Assignment Problem

The Generalized Assignment Problem (GAP) is an NP-hard optimization problem (Fisher et al., 1986). Let $t_{w,r}$ be the cost for agent $w \in \mathcal{W}$ to complete job $r \in \mathcal{R}$. Let $X_{w,r}$ be a binary decision variable, which is one if agent w is assigned to job r, and zero otherwise. Furthermore, let $a_{w,r}$ be the capacity absorption when assigning job r to agent w, and M_w is the maximum capacity of agent w. The objective is to minimize the total cost of assigning all jobs to agents. The problem can be formulated as follows:

$$\min \sum_{w \in \mathcal{W}} \sum_{r \in \mathcal{R}} t_{w,r} X_{w,r}, \tag{2.12}$$

subject to

$$\sum_{r \in \mathcal{R}} a_{w,r} X_{w,r} \leq M_w \quad \forall w \in \mathcal{W}, \tag{2.13}$$

$$\sum_{w \in \mathcal{W}} X_{w,r} = 1 \quad \forall r \in \mathcal{R}, \tag{2.14}$$

$$X_{w,r} \in \{0,1\} \quad \forall w \in \mathcal{W}, \forall r \in \mathcal{R}, \tag{2.15}$$

where (2.13) is to enforce the capacity constraints for each agent and (2.14) guarantees that all jobs are assigned each to one agent. In the context of assigning batches to pickers, each job is a batch, and an agent is a picker. Furthermore, in this thesis, $a_{w,r} = t_{w,r}$, i.e., the capacity absorption for an agent to complete a job is the same as the cost to complete it.

2.2 Algorithms

2.2.1 Dynamic Programming Algorithm for Optimal Batching

This algorithm searches for an optimal assignment of orders to batches. It is used to find optimal batching solutions for instances of limited size in sections 3.3 and 4.2. Let $\mathcal{O}$ be the set of orders to be processed, and N the maximum batch size, with $|\mathcal{O}|$ being the total number of orders. Each node in the algorithm is a pair composed of a state $\boldsymbol{a}$, indicating which orders are included in a solution, and the state's cost c_a. The algorithm operates using a priority queue-memory structure. As new nodes are added, the priority queue is sorted in ascending order by cost. Thus the lowest cost node is always on top. As the state, a binary vector $\boldsymbol{a} = \{a_1, ..., a_{|\mathcal{O}|}\}$, is used. If $a_i = 1$, order i is included in a node's solution, otherwise not. As the algorithm runs, and a new state $\boldsymbol{a}_{n+1}$ is reached, a map data structure m is used to keep track of the state of the entering state $\boldsymbol{a}_n$ so that $m(\boldsymbol{a}_{n+1}) = \boldsymbol{a_n}$. To find the (binary) batch r that was added to $\boldsymbol{a_n}$ to reach $\boldsymbol{a}_{n+1}$, the following formula is used: $r = \boldsymbol{a}_n \oplus \boldsymbol{a}_{n+1}$, where $\oplus$ is the binary exclusive-or operator. The steps of the algorithm are detailed below.

Step 0. Add a node with zero cost and state $\boldsymbol{a} = \{0, ..., 0\}$ to the priority queue.

Step 1. Read the top node p, and its state $\boldsymbol{a}$.

Step 2. If p contains all orders, extract the batch combinations in the found solution from m, return p and exit.

Step 3. Find the smallest index i in $\boldsymbol{a}$ that equals zero.

Step 4. Find the set of feasible batches $\mathcal{R}_i$ that include order no. i, but are not yet included in $\boldsymbol{a}$ (up to batch size of N).

Step 5. $\forall r \in \mathcal{R}_i$ generate new states $\boldsymbol{a}_r$.

Step 6. $\forall r \in \mathcal{R}_i$ find the costs $c_{\boldsymbol{a}_r}$ of $\boldsymbol{a}_r$.

Step 7. Remove p from the priority queue.

Step 8. $\forall r \in \mathcal{R}_i$ add new nodes $p_r = \{\boldsymbol{a}_r, c_{\boldsymbol{a}_r}\}$ to the priority queue, and set $m(\boldsymbol{a}_r) = \boldsymbol{a}$ Go to Step 1.

2.2.2 Clarke and Wright Based Algorithms

The order batching variants of the Clarke and Wright (1964) routing algorithm, C&W(i) and C&W(ii) (De Koster et al., 1999), are presented below. Both algorithms operate by updating a savings matrix, iteratively adding feasible pairs of orders or order clusters to the current solution. The savings matrices indicate the savings if two orders or groups of orders are batched together. Generally, the algorithms aim to add a pair of orders or order clusters with the maximum savings value in the matrix to the current solution, if feasible.

Basic variant: C&W(i)

This basic variant of the C&W algorithm provides a fast solution for the batching problem, which is very usable for initial solutions and subsequent local search. It requires the routing of all combinations of one and two customer orders from the set of all orders $\mathcal{O}$, which is usually computationally manageable (see Section 3.2.1). The algorithm consists of the following steps (De Koster et al., 1999):

Step 1. Calculate the savings for all possible customer order pairs $\{i, j\} \in \mathcal{O}$, where $\mathcal{O}$ is the set of all customer orders.

Step 2. Sort the savings in descending order.

Step 3. Select the pair with the highest savings.

Step 4. Three cases can be distinguished:

(i) Neither order has been previously included in the current solution s. Add a batch consisting of orders i and j to solution s.

(ii) Either i or j is included in s, but not both. Maximum batch capacity permitting, add the as-of-yet unincluded order to the batch of the included order. Otherwise proceed to Step 5.

(iii) Both orders have already been included in batches: proceed to Step 5.

Step 5. Select the order combination with the next highest savings, and proceed to Step 4.

If all customer orders $o \in \mathcal{O}$ are not yet present in solution s, add a single order batch for each remaining order. Finally, to get the total routing cost, the batches in the final solution are routed.

Recalculation of the Distance Matrix: C&W(ii)

The C&W(ii) algorithm considers order clusters as orders and replaces steps 4 and 5 with the following:

Step 4. Combine order clusters to form a new cluster, if allowed by the maximum batch size. If not, choose the next combination and repeat Step 4.

Step 5. If all orders have not been included in the current solution, proceed with Step 1, otherwise finish.

C&W(ii) is computationally much heavier than C&W(i), as it requires evaluation (and thus routing) of batches consisting of more than two customer orders inside the procedure to recalculate the distance matrix, instead of just routing batches as the algorithm finishes.

2.2.3 Variable Neighborhood Search

The VNS algorithm presented in Albareda-Sambola et al. (2009) uses three neighborhoods of different sizes to search a new solution in the neighborhood of solution $s \in \mathcal{S}$. The first neighborhood, $\mathcal{N}_1$, is composed of a single heuristic move, which moves one order from a batch to another. The second neighborhood, $\mathcal{N}_2$, is composed of $\mathcal{N}_1$ and two additional moves: one that transfers two orders from one batch to another, and one that transfers two orders to different batches. Neighborhood $\mathcal{N}_3$ includes $\mathcal{N}_2$ in addition to three other moves: the first swaps orders between two batches, the second transfers two orders from two separate batches to one other batch, and the third moves an order from batch a to batch b and an order from b to batch c. For a given instance I of a combinatorial problem, the VNS always reaches the same solution; there is no randomness involved in the search. See Figure 2.1 for an illustration: $r_a, r_b, r_c \in \mathcal{R}$ represent batches involved in each iteration of a search heuristic, $o_i, o_j \in \mathcal{O}$ the related orders, while the heuristics are denoted by $h_i \in \mathcal{H}$, where $i \in \{1, ..., 6\}$.

When the algorithm runs, there must exist at least one empty batch so that it is always possible to transfer an order from a batch. Batch capacity constraints are always maintained, and the feasibility of the solution is preserved throughout the search. The flow of the algorithm is detailed below in the following steps.

Step 0. Form an initial solution s_0 by including one batch per order in

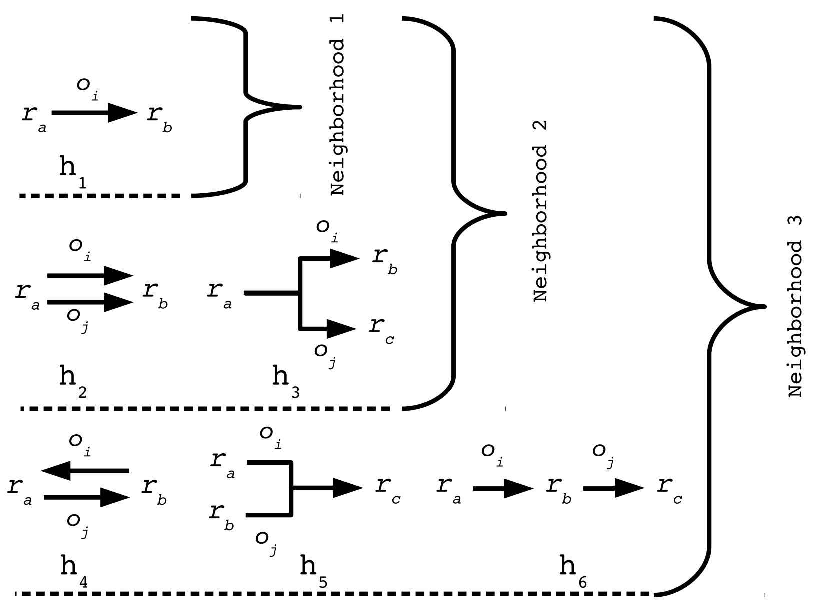


Figure 2.1. VNS neighborhoods and search heuristics. Adapted from Albareda-Sambola et al. (2009).

addition to an empty batch. Set $s = s_0$.

Step 1. Find the best neighboring solution s' in $\mathcal{N}_1(s)$. If $f(s') < f(s)$, set $s = s'$, repeat Step 1; otherwise go to Step 2.

Step 2. Compute the savings values for merging all batch combinations. If a solution with positive savings values exists among all combinations, find the solution s' where the merging of two batches will result in the biggest savings, set $s = s'$, and repeat Step 2. Otherwise proceed to Step 3. This step is similar to applying C&W(ii) to the batching problem.

Step 3. Find the best neighboring solution s' in $\mathcal{N}_1(s)$. If $f(s') < f(s)$, set $s = s'$, repeat Step 3; otherwise go to Step 4.

Step 4. Find the best neighboring solution s' in $\mathcal{N}_2(s)$. If $f(s') < f(s)$, set $s = s'$, repeat Step 4; otherwise go to Step 5.

Step 5. Find the best neighboring solution s' in $\mathcal{N}_3(s)$. If $f(s') < f(s)$, set $s = s'$, go to Step 4; otherwise exit.

To find the total routing cost, the sum of all costs of the batches batches present in s is calculated.

2.2.4 Attribute Based Hill Climber

The Attribute Based Hill Climber (Whittley and Smith, 2004) can be classified as a Tabu Search-type algorithm. Instead of using a classic Tabu-list of visited solutions which should be avoided, a memory of *attributes* and their corresponding values is maintained.

Let $s \in \mathcal{S}$ be the current solution, and $s' \in \mathcal{S}$ be the new candidate solution. For each problem, a set of neighborhood search heuristics (moves) is defined. Two sets of attributes can be distinguished by applying a move to reach solution s' from solution s: the entering $(s' - s)$ and the leaving $(s - s')$. The entering attributes form the set of attributes that do not exist in solution s but are present s', and vice versa for the leaving attributes.

Regardless of the type of combinatorial problem, the ABHC can be implemented with three algorithms, presented below (Whittley and Smith, 2004). Algorithm 1 returns a boolean value for the acceptance of candidate solution s', Algorithm 2 contains the attribute memory update function, and Algorithm 3 is a "steepest ascent/mildest descent" local search algorithm.

In Algorithm 1, line 1, a candidate solution s' is accepted if it contains an attribute for which a bigger value has previously been stored in memory. Otherwise if the new solution has a bigger cost (line 3), the new solution is rejected. Finally, if the cost of s' is higher than that of the biggest (worst) cost in s, it is rejected. Otherwise, if moving from solution s to s' deletes one of the worst attributes in solution s (lines 6-11), s' is accepted.

```
   Input: Solutions $s, s'$, attribute memory $amem$
   Output: Acceptance of solution $s'$
1  if $\exists a \in (s' - s)$ s.t. $f(s') < amem[a]$ then
2      return true;
3  else if $f(s') \geq f(s)$ then
4      return false;
5  else
6      for $i = 1$ to $|s - s'| + 1$ do
7          $a = worst(i, s)$;
8          if $f(s') \geq amem[a]$ then
9              return false;
10         else if $a \notin (s - s')$ then
11             return true;
12         end
13     end
14 end
```

Algorithm 1: ABHC acceptance

Algorithm 2 updates the attribute memory as needed only: if the cost of

s' is higher than that of s, only the new (entering) attribute values need to be updated (lines 1-2). Otherwise, all of the attributes are updated.

```
  Input: Solutions $s, s'$, attribute memory $amem$
1 if $f(s') \geq f(s)$ then
2     $amem[a] = \min(amem[a], f(s')) \quad \forall a \in (s' - s)$;
3 else
4     $amem[a] = \min(amem[a], f(s')) \quad \forall a \in s'$;
5 end
```

Algorithm 2: ABHC memory update

A single local search iteration of the ABHC is described in Algorithm 3. On line 1, the worst $|s - s'| + 1$ attributes in s are identified and stored. The cost of the current best solution (*best*) is initialized infinity on line 2. The neighborhood of s is iterated through updating the best solution when a better solution is found and accepted (lines 3-7). Finally, updates to the attribute memory are done on line 9 if a new candidate solution is found in $\mathcal{N}(s)$.

```
   Input: Current solution $s$, attribute memory $amem$
   Output: Updated solution $s$
1  identifyWorstAttributes($s$)
2  $best = null$ {initialize $f(best) = \infty$}
3  for $all\ s' \in \mathcal{N}(s)$ do
4      if $f(s') < f(best)$ and $accepts(s', s)$ then
5          $best = s'$
6      end
7  end
8  if $best \neq null$ then
9      update($s, best$)
10     $s = best$
11 end
```

Algorithm 3: ABHC main routine

Henn and Wäscher (2012) use ABHC to solve the order batching problem. Two types of attributes are explored: the presence of a pair of orders $\{o_1, o_2\}$ in any batch, and the assignment of an order o to a batch r (i.e., the pair $\{o, r\}$). They find that using the pair of orders as the attribute leads to better performance in their algorithm. Two deterministic heuristics form the neighborhood of a solution s: a swap of two orders between batches and a move of an order from one batch to another (h_4 and h_1 in Figure 2.1, respectively).

2.2.5 Large Neighborhood Search

According to Pisinger and Ropke (2010), neighborhoods in LNS (Shaw, 1997) methods are defined by destroy and repair methods. They are also called in removal and insertion methods in other sources, e.g., in Ropke and Pisinger (2006a) and Gharehgozli et al. (2013). The destroy method of a neighborhood search heuristic $h \in \mathcal{H}$ breaks down a part of a solution s in a predefined manner, while the repair methods try to find good alternatives to s by constructing another feasible solution s' in the neighborhood of s based on the repair method associated with h. These methods typically contain some element of randomness in choosing what parts of s to modify.

Let $i(s')$ denote a repair and $e(s)$ destroy function of a LNS algorithm for a partial solution $s' \notin \mathcal{S}$ and solution $s \in \mathcal{S}$, respectively. Algorithm 4 shows the pseudocode implementation of a LNS algorithm. As input, a feasible starting solution is needed. The algorithm iteratively applies destroy and repair methods in succession, and it stores new best known solutions until an exit criterion is met (such as a cap on the number of iterations). The acceptance function on line 4 can take the form of a greedy heuristic or something else, e.g., a simulated annealing acceptance criterion (see Section 6.3.4).

```
Input: Feasible solution $s$
Output: Best found solution $s$
1  $s_{best} = s$
2  while Stop criterion is not met do
3      $s' = i(e(s))$
4      if accept($s, s'$) then
5          $s = s'$
6      end
7      if $f(s') < f(s_{best})$ then
8          $s_{best} = s'$
9      end
10 end
11 return $s_{best}$
```

Algorithm 4: Large Neighborhood Search (Pisinger and Ropke, 2010)

2.2.6 Adaptive Large Neighborhood Search

The probability of choosing heuristic $h \in \mathcal{H}$ with the roulette wheel method is

$$q_h = \frac{\theta_h}{\sum_{h' \in \mathcal{H}} \theta_{h'}}, \tag{2.16}$$

where $\theta_h \in \mathbb{R}^+$ is the *weight* of heuristic h. As the algorithm runs, some heuristics will perform better than others. Good performance of a heuristic will increase its weight and thus the probability of it being selected again by (2.16). Each of the heuristics has a *score*, which is updated whenever the heuristic is selected based on the heuristic's search success. Each score Ψ_h is updated by:

$$\Psi_h = \Psi_h + \begin{cases} \sigma_1 & \text{if the new solution is the global best.} \\ \sigma_2 & \text{if the new solution is better than the current one.} \\ \sigma_3 & \text{if the new solution is accepted.} \end{cases} \tag{2.17}$$

An update to a heuristic's weight θ_h is done every δ iterations by

$$\theta_h = \lambda/\zeta_h\theta_h + (1-\lambda)\Psi_h, \tag{2.18}$$

where λ is the *decay* parameter, which controls the rate of change in the weights, and $\zeta_h \geq 1$ is a normalization factor which reflects the computational effort that h requires (Ropke and Pisinger, 2006b). At the beginning and after an update of the scores, i.e., every δ iterations, the values Ψ_h are set to zero $\forall h \in \mathcal{H}$. If a hill climbing scheme is not used in the algorithm updates to the scores occur only when a new global best solution is found, i.e., Ψ_h can only get the value σ_1.

A pseudocode for the ALNS is shown in Algorithm 5 (Pisinger and Ropke, 2010). The main differences between algorithms 5 and 4 are the inclusion and updates of the heuristics' selection probabilities on lines 2 and 12, respectively. Additionally, instead of explicitly referring to each pair of destroy and repair methods, their combinations are marked more generally with h.

2.3 Summary

Each of the presented combinatorial optimization problems is NP-hard by itself. Combining this with the realistic instance sizes dealt with in this thesis, which in most cases result in very large search spaces and/or long computational times, local non-optimal methods are in practice the only option to solve the problems. The presented algorithms, apart from the C&W methods and the optimal batching algorithm, can be classified as local search methods. All the other algorithms are deterministic in

```
Input: Feasible solution $s$
Output: Best found solution $s$
1  $s_{best} = s$
2  $p_h = \{1, ..., 1\}$
3  while Stop criterion is not met do
4      select neighborhood search heuristic $h \in \mathcal{H}$ using $p_h$
5      $s' = h(s)$
6      if accept($s, s'$) then
7          $s = s'$
8      end
9      if $f(s') < f(s_{best})$ then
10         $s_{best} = s'$
11     end
12     update $p_h$
13 end
14 return $s_{best}$
```

Algorithm 5: Adaptive Large Neighborhood Search (Pisinger and Ropke, 2010)

the sense that their exploration does not depend on any random routine. For reasonably sized problems with no computationally expensive subprocedures that must be run when new solutions are found, they function very well, as can be seen from the following chapters. However, when search spaces grow in size, such as when considering two of the presented problems in unison, or if they have expensive subprocedures, deterministic search algorithms cannot explore the space of solutions in reasonable time.

3. Order Batching when Items Have to Be Picked in a Strict Sequence

Batching orders and routing pickers while respecting precedence constraints is complex. Order batching is NP-hard for batches of size three or higher (Gademann and Van de Velde, 2005). Routing pickers while respecting precedence constraints has complexity of $O(N^2(P+1)^N)$ (Psaraftis, 1980a), where N is the number of customer orders in the batch and P is the maximum number of items in an order. Finding exact solutions for large orders and large batches becomes rapidly intractable. The company used as a reference in this thesis has 2,000 orders per day and up to 50 products per order, which have to be batched in batches of three orders. Batching aims to reduce total travel distance by combining multiple similar orders (which have to visit overlapping parts of the warehouse). As the problem has to be solved multiple times per day and because of large and exponentially growing search spaces and relatively time consuming routing exact computation of the order batching is not feasible and thus a heuristic method is used.

The problem is solved using a generic solution method. For the batching, a simulated annealing-based combinatorial search algorithm based on maximizing total savings in travel distance is used. Selecting the orders to be batched is based on comparing all pairs of orders and estimating the savings due to combining a larger number of orders. This reduces computation times substantially, while the error compared to optimal combinations is within 1.2%. The routing can be solved using an A^*-type shortest path algorithm. The method is exact and general, not constrained by the particular warehouse layout. The method is readily applicable to any warehouse where batch picking is applied in waves and the orders have strict internal precedence constraints.

The contribution of the research presented in this chapter is a structured approach for the joint order batching and picker routing problem

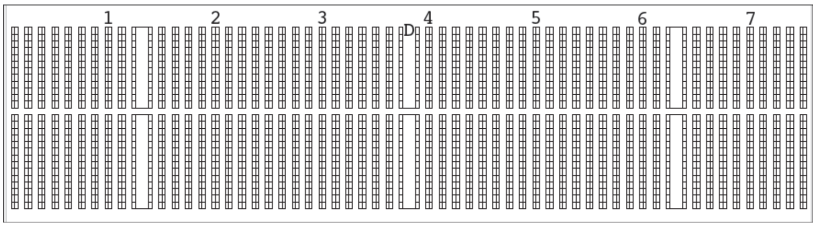


Figure 3.1. Top view of the warehouse with seven drop-off locations indicated at the top (marked 1-7). The empty bin depot is denoted with D. The 57 aisles are unidirectional, while the three cross-aisles are bidirectional. The aisle-to-aisle distance is 5.5 m the slot-to-slot distance is 3.7 m, which is also the width of the central cross-aisle.

in warehouses with any layout or any strict sequence constraint using an innovative way of estimating batch savings. The method is fast, and compares well to optimal solutions and heuristics from literature. In the reference case, travel distance savings of nearly 16% could be achieved.

3.1 Problem Description

In this section the combined batching and routing problem as a variant of the precedence-constrained Traveling Salesman Problem is described. The process is as follows. A given number of trucks pick the customer orders. Each customer order represents a roll container (a *bin* in this context) to which items must picked to in a strict pre-specified sequence. The items do not have to be stacked on top of each other in this order, if there is space in the bin to do otherwise. Each truck starts completely empty and then receives a set of orders to be picked. The number of received orders per truck does not exceed N, the number of bins carried by a truck.

Before the items of an order can be picked, for each order an empty bin has to be collected at the empty bin depot (see Figure 3.1). Subsequently, each truck travels through the warehouse to pick the items of the orders in the pre-specified sequence per order. Once all items of an order have been picked, the truck delivers the order at its drop-off location. Once all orders have been completed and the truck is empty again, it receives a new batch of orders. A batch is the set of orders handled in a route by a truck between the moment the truck receives this set and the moment it has dropped off *all* orders and is empty again.

Given the assignment of orders to batches, the sequence in which the batches are carried out is not relevant for the total picking time.

3.1.1 Notation

There is a set $\mathcal{O}$ of orders to be carried out. Each order consists of a number of items that have to be picked at specific pick locations. All items of an order are picked in a roll container of one of the available trucks (see Figure 1.2). Once all items of an order have been picked, it must be delivered at its pre-specified drop-off location (one of the locations marked with 1-7 in Figure 3.1).

The set of all ordered items is indicated by $\mathcal{G}$. Every item $g \in \mathcal{G}$ is characterized by three attributes: pick location; the customer order to which it belongs to; and the number of the units that need to be picked. The order to which item g belongs is indicated by $o(g) \in \mathcal{O}$. The items of each order must be picked in a pre-specified sequence. The sequence number of item $g \in \mathcal{G}$ of order $o(g) \in \mathcal{O}$ is indicated by p_g. The distance between items g and $k \in \mathcal{G}$ is equivalent with the distance between the corresponding pick locations. This distance is denoted by $d_{g,k}$.

The first item of an order corresponds with picking up an empty roll container at the empty roll container depot. The last item of an order corresponds with the delivery of the order at its drop-off location. The sets of first and last items are denoted by $\mathcal{G}_f$ and $\mathcal{G}_l$, respectively.

Note that the model described in this section is only meant to explain the combined batching and precedence-constrained routing problems in detail. For computational purposes other models and techniques that are described later on in this chapter are used.

Model In order to model the batching and routing problem, (i) the orders have to assigned to the batches, and (ii) the sequence of the items of each order needs to be kept track of. To that end, a set $\mathcal{P}$ of potential batches is introduced. The number of potential batches should be sufficient to carry out all orders. For example, $|\mathcal{P}| = |\mathcal{O}|$ is certainly enough (but often much too large), since this corresponds to the situation that each order is picked on its own. Each batch can contain up to N bins. The set of bins is denoted by $\mathcal{B}$. The batch to which bin $b \in \mathcal{B}$ belongs is denoted by $r(b) \in \mathcal{P}$.

For each order $o \in \mathcal{O}$ and bin $b \in \mathcal{B}$ a binary decision variable $Y_{o,b}$ is introduced to indicate whether or not order o is picked in bin b. Furthermore, let $X_{g,k,r}$ be a binary decision variable indicating whether batch $r \in \mathcal{P}$ goes directly from item $g \in \mathcal{G}$ to item $k \in \mathcal{G}$. Finally, let $U_g \in \mathbb{N}$ indicate the relative position of item $g \in \mathcal{G}$ in the tour that picks the order of item g. The relative position is the integer position of g in the tour.

Now the model can be described as follows:

$$\min \sum_{g\in\mathcal{G}} \sum_{k\in\mathcal{G}\setminus\{g\}} d_{g,k} \sum_{r\in\mathcal{P}} X_{g,k,r} \tag{3.1}$$

subject to

$$\sum_{b\in\mathcal{B}} Y_{o,b} = 1 \qquad \forall o\in\mathcal{O} \tag{3.2}$$

$$\sum_{o\in\mathcal{O}} Y_{o,b} \leq 1 \qquad \forall b\in\mathcal{B} \tag{3.3}$$

$$\sum_{k\in\mathcal{G}\setminus\{g\}} X_{g,k,r} = \sum_{b\in\mathcal{B}:r(b)=r} Y_{o(g),b} \qquad \forall g\in\mathcal{G}\setminus\mathcal{G}_l \ \ \forall r\in\mathcal{P} \tag{3.4}$$

$$\sum_{g\in\mathcal{G}\setminus\{k\}} X_{g,k,r} = \sum_{b\in\mathcal{B}:r(b)=r} Y_{o(k),b} \qquad \forall k\in\mathcal{G}\setminus\mathcal{G}_f \ \ \forall r\in\mathcal{P} \tag{3.5}$$

$$U_g - U_k + |\mathcal{G}|\cdot\sum_{r\in R} X_{g,k,r} \leq |\mathcal{G}|-1 \qquad \forall g,k\in\mathcal{G}: g\neq k \tag{3.6}$$

$$U_g - U_k \geq p_g - p_h \qquad \forall g,k\in\mathcal{G}: o(g)=o(k)\wedge p_g > p_k \tag{3.7}$$

$$Y_{o,b}\in\{0,1\} \qquad \forall o\in\mathcal{O} \ \ \forall b\in\mathcal{B} \tag{3.8}$$

$$X_{g,k,r}\in\{0,1\} \qquad \forall g,k\in\mathcal{G}: g\neq k \ \ \forall r\in\mathcal{P} \tag{3.9}$$

$$U_g\in\mathbb{N} \qquad \forall g\in\mathcal{G} \tag{3.10}$$

According to (3.2), each order is assigned to exactly one bin, and (3.3) states that each bin covers at most one order. Thus the orders are correctly assigned to the bins.

According to (3.4), each location related to an item (except a drop-off) is exited so that the corresponding order has been assigned to one of the bins in the batch. Similarly, constraints (3.5) state that each item, except a first location of an order, is entered so that the corresponding order is assigned to one of the bins of the batch. It follows that all items of an order are picked by the same batch.

Furthermore, constraints (3.6) are based on the TSP subtour elimination constraints of (Miller et al., 1960). According to (3.6), the values U_g represent the (relative) positions of the items in each batch: if a batch goes directly from item g to item k, then (3.6) implies $U_k \geq U_g + 1$. The values U_g need not be consecutive integers within a batch. Anyway, for two items g and k of the same batch, $U_g < U_k$ stands if and only if item g is picked before item k. It also follows that the sequence of items of a batch does not contain subtours.

Furthermore, constraints (3.7) state that two items belonging to the same order are picked by the same batch in such a way that their (relative) positions in the batch are at least the a priori specified difference. Thus $U_g < U_k$ if and only if $p_g < p_k$. It follows that the precedence constraints of the items of each order are satisfied.

Constraints (3.8), (3.9) and (3.10) specify the binary and non-negative character of the decision variables.

The objective function (3.1) represents the total routing costs of the batches. Thus the optimal solution of (3.1)–(3.10) is the solution with minimum total routing costs.

3.2 Solution Approach

The algorithm presented in this section, Precedence-Constrained Estimated Savings based batching, or PCES for short, incorporates two distinct sub-algorithms:

1. an optimal A^*-algorithm for the routing;

2. a simulated annealing-based combinatorial search algorithm, SA-REMIX, for searching the space of possible batches.

Figure 3.2 shows the flow diagram of PCES. The initialization is performed by inputting a selected set of customer orders $\mathcal{O}$, in step 1, usually more than can be processed at once. In step 2, the next set of R customer orders is selected for batching. Then batches with two customer orders are constructed and routed in steps 3 and 4. In step 5, estimates of savings for batches larger than two are calculated. To find the best batch allocation, a combinatorial search is performed in the space of batches with real and estimated savings in step 6. Estimated savings are calculated as needed in steps 5 and 6. In step 7, optimal routes are found for those batches that maximize the savings. Steps 5 and 6 are repeated for a fixed number of iterations to find a good solution for the order-batching problem. If some customer orders are left in $\mathcal{O}$, the algorithm returns to step 2, otherwise it terminates with its solution in step 8.

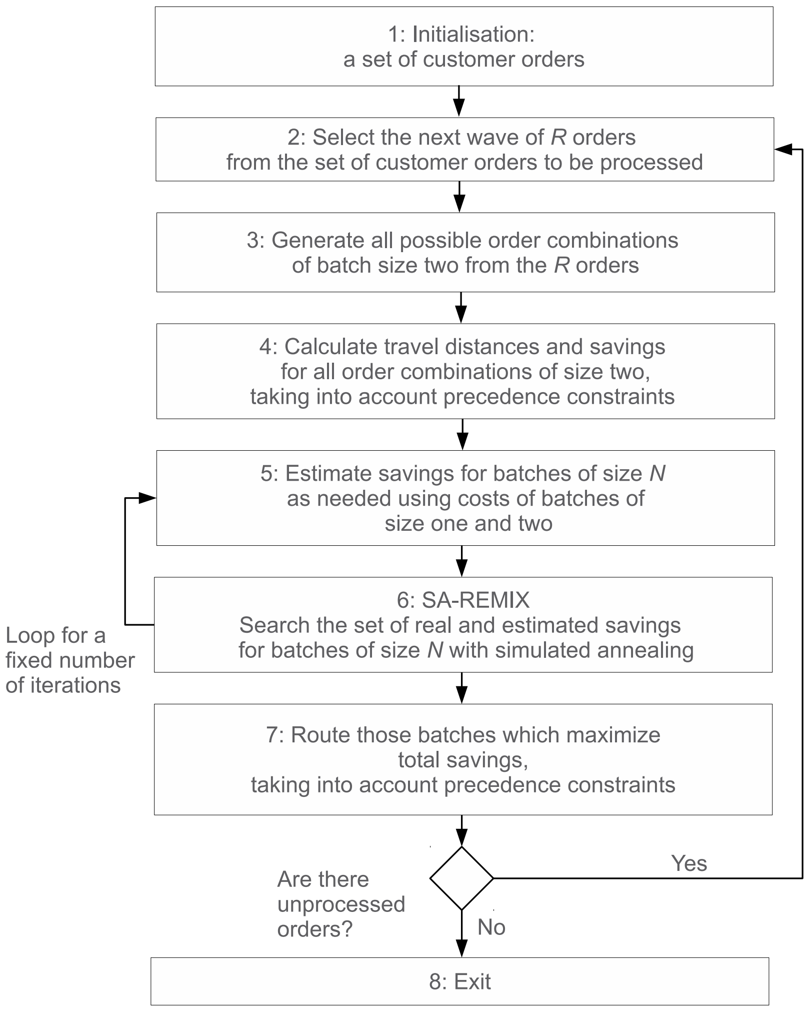

1: Initialisation:
a set of customer orders
2: Select the next wave of R orders
from the set of customer orders to be processed
3: Generate all possible order combinations
of batch size two from the R orders
4: Calculate travel distances and savings
for all order combinations of size two,
taking into account precedence constraints
5: Estimate savings for batches of size N
as needed using costs of batches of
size one and two
6: SA-REMIX
Search the set of real and estimated savings
for batches of size N with simulated annealing
Loop for a
fixed number
of iterations
7: Route those batches which maximize
total savings,
taking into account precedence constraints
Are there
unprocessed
orders?
Yes
No
8: Exit

Figure 3.2. Solution approach for solving the batching problem with estimated savings with PCES.

3.2.1 Estimating Savings from Batching More than Two Customer Orders

The main reason for batching customer orders is to generate savings in travel distance compared to picking these customer orders independently.

If computationally feasible, all possible routes can explicitly and exhaustively be constructed from all possible customer order combinations and a combinatorial search performed in the exhaustive space, thus minimizing the travel distance. The number of possible combinations C up to the maximum batch size N of all R customer orders can be calculated by

Equation (6.9), below.

$$C = \sum_{i=1}^{N} \binom{R}{i} \tag{3.11}$$

subject to $R \geq N$ and $C, R, N \in \mathbb{N}$.

Especially with larger batch sizes ($N \geq 3$), explicitly evaluating all possible batch combinations quickly becomes intractable due to the large number of combinations and the complexity of the routing. This is countered by estimating the savings made for batches of more than two customer orders.

Instead of performing a combinatorial search in the explicitly evaluated search space of all batch travel distances, only a part of the space consisting of batches each with real or estimated savings is searched. Real savings are found by constructing routes for batches of size two and subtracting their travel distances from the respective single-order travel distances. Estimates for savings of batches of size three or larger are calculated on the basis of the real travel distances of batches of sizes one and two. With PCES based batching, in addition to routing all batches of size two and one, only those batches with three or more customer orders which are found using SA-REMIX need to be routed. Compared to routing all batches with three or more orders, the number of calls to the routing algorithm for batches of size three or more is reduced from

$$C_{3+} = \sum_{i=3}^{N} \binom{R}{i} \tag{3.12}$$

to R/N (plus a possible remainder if $N > 3$).

In the following the way to estimate the travel savings for batches of three and more orders is explained. Let i, j and k be three orders that can be batched together. If by batching order i with j, some travel distance is saved when compared to picking them independently, batching is beneficial. The savings resulting from batching the two customer orders i and j can be calculated by:

$$S_{i,j} = c_i + c_j - c_{i,j} \tag{3.13}$$

where $c_{i,j}$ is the batched travel distance of customer orders i and j, c_i and c_j the travel distances for picking customer orders i and j independently. The travel distances in Equation (3.13) are the real travel distances which have been evaluated with the routing algorithm presented in Section 3.2.2.

Likewise, let batching *i* with *k* and *j* with *k* be beneficial. This would then imply that batching all three of these customer orders into a batch of size three would also be beneficial. If these three customer orders are batched, the savings can be calculated by:

$$S_{i,j,k} = c_i + c_j + c_k - c_{i,j,k} \tag{3.14}$$

Expanding from this, if *i, j* and *k* are batched, the savings can be estimated by:

$$\bar{S}_{i,j,k} = S_{i,j} + S_{i,k} + S_{j,k} \tag{3.15}$$

Double counting can be eliminated by dividing (3.15) by two. However, as these three order batch savings estimates are used in the same search space with the real two order batch savings, an added emphasis is given to the larger batches by not eliminating the double counting. By combining (3.13) and (3.15), the estimated savings of a size three batch can be calculated:

$$\bar{S}_{i,j,k} = 2c_i + 2c_j + 2c_k - c_{i,j} - c_{i,k} - c_{j,k} \tag{3.16}$$

Let $\mathcal{V}$ be a set of n arbitrary customer orders $\{1...n\}$ and c_n be the travel distance of customer order n. Using this notation, if a batch is formed from the customer orders in $\mathcal{V}$, the savings can be found by:

$$S_{\mathcal{V}} = \sum_{i \in \mathcal{V}} c_i - c_{\mathcal{V}} \tag{3.17}$$

Using the same notation as above, the total savings resulting from batching N customer orders can be estimated:

$$\bar{S}_{\mathcal{V}} = \sum_{i=1}^{N-1} (\sum_{j=i+1}^{N} (c_i + c_j - c_{i,j})) \tag{3.18}$$

Using (3.18), estimates can be calculated for all possible combinations up to a given batch size. To use this equation, only the batch tour distances of batches of size one and two are needed. The real savings for batches of size two are calculated, while for size three and more they are estimated. For batches of size one, the savings value is set to zero. In practice, estimates are calculated as they are needed, i.e., in steps 5 and 6 of Figure 3.2.

3.2.2 Pick Route Construction

The A^*-algorithm introduced by Hart et al. (1968) is used to find the optimal pick route. This algorithm's states are organised in a lattice, as states can be reached through multiple paths. The A^*-algorithm's state is defined by the vector $\mathbf{m} = (m_0, m_1, ..., m_N)$, similar to Psaraftis (1980b) and Psaraftis (1983). $m_0 \in \{1, ..., N\}$ defines the customer order in which the last product was placed. m_i, $i \in \{1, ..., N\}$, is the number of locations visited for customer order i (including the delivery location).

An example is shown in Figure 3.3. The algorithm's state starts with empty bins, and the picker at the depot. The state in the starting location is $\mathbf{m}_1 = (m_0, 0, ..., 0)$, where m_0 is an arbitrary customer order. To pick an item to customer order i, m_i is incremented by one and set $m_0 = i$. Keeping track of m_0 is important, as the decision of which of the orders to pick next depends on the location of the storage slot of the item that was last picked. Otherwise there would be up to N possible locations where the picker could be when calculating distances to the next possible pick locations instead of just one. Figure 3.4 gives an example of the state space for two customer orders with two items to be picked in each of them. The arcs between two adjacent states represent possible state transitions by moving to the next location and picking the item required. The A^*-algorithm searches the lattice by using the exact travel distance to reach a state plus an estimate for the distance to reach the final state, where all customer orders have been picked and delivered and the picker is at the depot. As long as the estimate underestimates the remaining travel distance, the algorithm is guaranteed to find the optimal solution as it progresses through the lattice (Hart et al., 1968). Psaraftis (1980b) uses a dynamic programming approach with no estimated distances to prune the search space. He also considers additional constraints which are not used here, such as customer preferences between waiting and riding, and time windows (Psaraftis, 1983).

The shortest total travel distance to reach a state $\mathbf{m}$ for a given set of customer orders $\mathcal{V}$ is denoted by $d(\mathbf{m}, \mathcal{V})$. For each state and the selected set of customer orders, an estimate for the remaining travel distance from state $\mathbf{m}$ to the depot and add it to the state cost is calculated by $\bar{d}(\mathbf{m}, \mathcal{V})$. The single customer order with the longest remaining travel distance to the goal is found and $\bar{d}(\mathbf{m}, \mathcal{V})$ is set to equal that distance. Let $\bar{d}_i(\mathbf{m}, \mathcal{V}_i)$ be the travel distance to goal by only visiting the locations of customer order

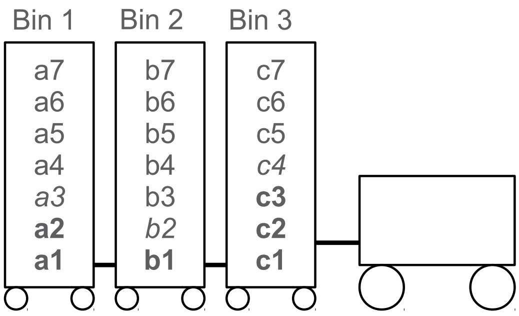


Figure 3.3. A truck picking three customer orders into three separate bins. Items in bold have already been picked; the underlined one denotes the last pick and italicized ones are possibilities for the next transition, representing state (2,2,1,3).

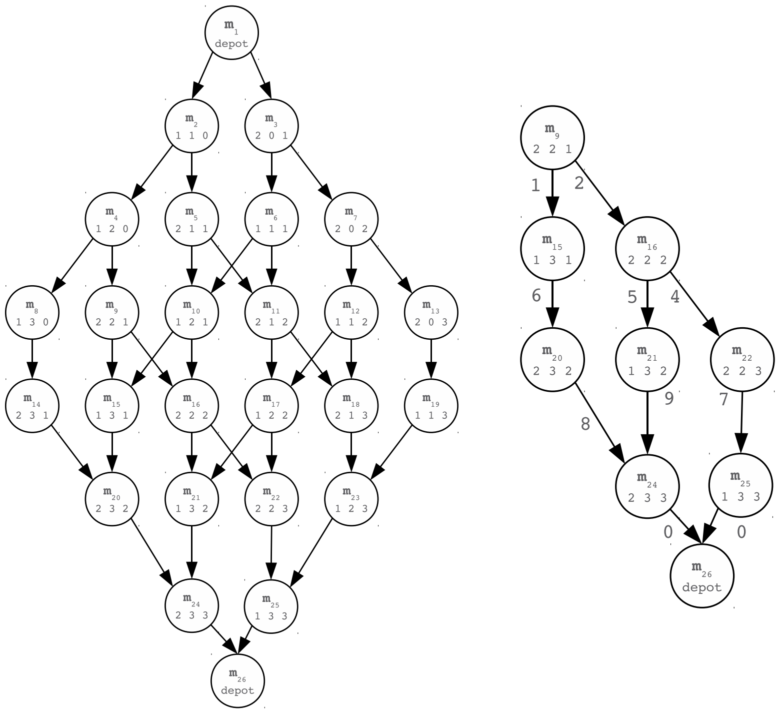


Figure 3.4. Left: a Hasse diagram of the state space formed by two customer orders with two items to be picked into each of them. Inside the circles, the state name is at the top, the state vector is at the bottom. The first member of the state vector refers to the customer order where the last pick occurred, the following two refer to the customer order location indices. The highest location index (3 in this case) represents the delivery location for the given customer order. Right: possible transitions and distances from the state $\mathbf{m}_9$ in the Hasse diagram.

i, $i \in \{1, ..., N\}$. The estimate is calculated by:

$$\bar{d}(\mathbf{m}, \mathcal{V}) = \max_{i \in \{1,...,N\}} \bar{d}_i(\mathbf{m}, \mathcal{V}_i). \tag{3.19}$$

The estimate of the total distance to the goal state is calculated from $\bar{e}(\mathbf{m}, \mathcal{V}) = d(\mathbf{m}, \mathcal{V}) + \bar{d}(\mathbf{m}, \mathcal{V})$. Doing this prevents some of the unnecessary exploration of states that are not part of the optimal tour.

As an example, if there are two customer orders with two products to be picked in each and a delivery location for both orders, a lattice is formed by all the possible states $\mathbf{m}_i$, $\forall i \in \{1, ..., 26\}$. In Figure 3.4, all states are enumerated with $\mathbf{m}_1$ being the starting state and $\mathbf{m}_{26}$ being the end state. Arrows indicate the reachability of states. The number of states in the lattice and the time complexity for the algorithm (Psaraftis, 1980a) can be found from (3.20) and (3.21), respectively. Note that in (3.21) the product is preceded by N^2, and not by N as in (3.20). This is due to the fact that all states in the lattice have at most N incoming arcs (see Figure 3.4).

$$M = O(N \prod_{i=1}^{N} |\mathcal{V}_i + 1|), \tag{3.20}$$

$$V = O(N^2 \prod_{i=1}^{N} |\mathcal{V}_i + 1|). \tag{3.21}$$

where the sum term removes all states that are infeasible such as $[1, 0, 1]$. $|\mathcal{V}_i|$ is incremented with one as the delivery location is included.

In Figure 3.4 on the right side, the possible transitions and distances from state $\mathbf{m}_9$ are shown. Let the total distance to that state be a_9. To calculate the heuristic distance to goal, all remaining items can either be added to bin 1 or 2, but not to both. The heuristic distances are thus $\bar{d}_1(\mathbf{m}_9, \mathcal{V}) = 1$ (as only one item can be added to order 1) and $\bar{d}_2(\mathbf{m}_9, \mathcal{V}) = 2 + 4 = 6$ (as two items can be added to order 2). $\bar{d}_2(\mathbf{m}_9, \mathcal{V})$ is chosen for the heuristic as per (3.19). Then the estimate of the total cost of the goal state is $\bar{e}(\mathbf{m}_9, \mathcal{V}) = a_9 + \bar{d}_2(\mathbf{m}_9, \mathcal{V})$. The estimate for the total distance to goal is very optimistic. If a better estimate were to be used, the running times of the algorithm might decrease.

The A^*-algorithm is used to route batches of two customer orders, as well as the final batches after batching has been finished, in steps 4 and 7 of the PCES algorithm, as explained in Figure 3.2, respectively.

3.2.3 Batching

After all savings and travel distances for batches of size two have been determined, orders have have to be batched in batches of size up to N. The batching algorithm presented in this section, SA-REMIX, is a Large Neighborhood Search algorithm (see Section 1.4.5) consisting of a construction phase followed by improvements embedded in a simulated annealing scheme to avoid local optima. The decision to use a simulated annealing metaheuristic in the algorithm was made based on it being previously incorporated into Large Neighborhood Search-type methods in Ropke and Pisinger (2006a) and Pisinger and Ropke (2010). C&W(i) is used to form an initial solution. This starting solution gives an upper bound for the total travel distance. Next, a REMIX (a random destroy-random repair) procedure is applied.

REMIX This procedure randomly selects a fixed number of batches from the ones that are currently selected (steps 5 and 6 in Figure 3.2). The orders contained in the batches are mixed and randomly sampled again to form a number of new batches. The REMIX procedure is embedded in a simulated annealing routine. The size of the new batches is limited to N. The total travel distance savings of these new batches (S_{new}) are estimated and compared with those of the original batches (S_{org}). Let $Q \sim U[0,1]$ be a uniformly distributed random variable and T_n be the current temperature, where n is the current iteration. New batches are accepted if one of the following two criteria is met:

1. if their savings are larger than those of the original batches, i.e., $S_{new} > S_{org}$;

2. or, otherwise, if $Q < \exp((S_{new} - S_{org})/T_n)$.

The initial temperature T_0 is set to equal the highest savings value of the batches with two customer orders. Updates to T_n are done using *Lundy's* (Lundy and Mees, 1986) temperature schedule: $T_{n+1} = T_n/(1 + T_n/n)$.

3.3 Validation and Results

In this section, the results of batching with the method presented in Section 3.2.3 are compared both with company results and with optimal and heuristic batching, while using the A^*-routing. The company data is used as input for the experiments. Data consisting of voice-picking logs of almost 35,000 batches (approximately 100,000 customer orders) was analyzed, each consisting of up to three customer orders and an approximate total of 1.8 million log entries (i.e. pick locations to be visited), spanning the first three months of 2011. Each log entry includes a slot address, pick time, batch number, customer order number and picker number. Also the sequence in which each customer order should be picked is given. Currently the batches are created by grouping the customer orders per truck delivery route in a FCFS sequence. The pick routes are constructed using the S-shape heuristic, preserving single-directional aisle travel and strict item precedence.

The experiments are carried out with 1,536 customer orders sampled from the data set (about one day's work). The customer orders of each day are grouped in waves. Each wave represents a number of truck delivery routes with scheduled departure times within a certain time interval. To make a meaningful comparison between different algorithms, the number of customer orders per wave, or wave size R, is set to vary between 12 and 150. As most of the orders arrive early before the morning shift starts, it is possible to do order batching for much larger instances, especially if truck departure times overlap. Due to the relatively long computational time of the routing, such instances are impractical to solve with most of the tested algorithms from the literature. The results show, that after the wave size of 126 the differences in relative performance between the different algorithms do not change much, only the computational time grows.

All experiments were run single-threaded on an Intel(R) Core(TM) i7 920 CPU running at 2.67 GHz and with 12 GB of memory. Due to the exponential complexity of most algorithms that solve the order batching problem for a maximum batch size of three or more orders, the time to solve a given problem instance is considered in addition to the solution quality. The space of possible batch combinations when maximum batch size is three or more increases exponentially with the wavesize R. Thus the number of simulated annealing iterations run with PCES is increased with wavesize. A good number of iterations for the SA-routine that scales with both wave and batch sizes was found to be $J = 500,000 + (2N - 5)R^2$. This was determined ad-hoc fashion. Other amounts should work as well, but this amount produced consistently good results and proved scalable.

Case Description The warehouse is operated by a retail store chain selling non-food products and supplying 148 stores mostly on a daily basis. Orders arrive online and can be processed in five or six waves per day of R customer orders each. Orders are picked manually from pallet racks into customer orders using a picker-to-parts order picking system (De Koster et al., 2007). Pickers have to travel long distances as the warehouse has 57 aisles of 104 meters each and three cross-aisles, see Figure 3.1. The total warehouse area is 75,000 m^2, the average pick tour is 880 meters, and a warehouse picker travels an average of 7300 meters a day. The company sells and stores over 50,000 SKUs. Each warehouse aisle has 52 pick slots, which are divided into four subslots and two storage levels. This amounts to a total of 23,296 different pick locations in the warehouse. These locations are condensed into 2912 locations and corresponding items. The distance between two locations is measured using Manhattan distances, taking into account that pickers can only travel along the aisles in one direction, for safety reasons, and to avoid congestion. If the next pick is from a slot which is in the same aisle but in opposite travel direction, the travel distance is calculated respecting these traffic rules.

The warehouse uses a family grouping strategy to allocate items to storage slots, similar to the stores. Such a storage strategy is common in many retail warehouses. Efficiency losses in the warehouse by not using class-based storage are traded-off against efficiency gains in the stores by faster customer order unloading and shelf replenishment (De Koster et al., 2007). The customer orders must be filled in a predefined sequence, primarily due to the family grouping. Each item has a unique pick location in the warehouse. The contents and pick sequences for each of the customer orders are generated a priori by a Warehouse Management System (WMS). Every pick route contains up to three customer orders (a batch), currently all for the same store, in one tour of the warehouse (see Figure 3.1), but this might change, depending on potential travel time savings. A picker drives an order picking truck (see Figure 1.2) and follows a tour determined by the on-board customer orders, directed by a truck-mounted RF terminal. In addition to the truck terminals, the company uses a pick-by-voice system, which is not used for guiding the pickers, but only for pick confirmation (De Koster et al., 2007). Pickers can overtake each other. The warehouse layout can be seen in Figure 3.1. The delivery location of each completed customer order is known in advance: it is the one closest to the dock door from which the outbound truck will depart to the store.

Each order picking tour related to a single batch starts from the depot, see Figure 3.1. Before returning to the depot, all relevant pick locations and drop-off points must be visited. There are seven drop-off locations in the warehouse.

Validation of the Estimation Method To speed up batch evaluation, all batches of three customer orders or more are not explicitly routed in PCES but estimated using (3.16) or (3.18). To validate this approach, a set of 10,000 batches ($\mathcal{P}$) of three customer orders is sampled from the previously sampled data set of 1,536 customer orders. The 10,000 batches are all routed, and real and estimated savings are calculated for all $r \in \mathcal{P}$ using (3.14) and (3.16), respectively. Thus when calculating real savings, explicit routing is done for batches size three or more as well. All batches with negative estimated savings are discarded, as they would never be chosen by SA-REMIX, resulting in just over 5000 batches or the set $\mathcal{P}^+ \subset \mathcal{P}$. Both sets of savings, i.e., real and estimated ($\mathcal{P}_v^+$ and $\mathcal{P}_e^+$, respectively), are then ordered in descending order to establish real and estimated integer batch ranks v_v^r and v_e^r, $\forall r \in \mathcal{P}^+$. For validation of the estimate of a batch r, a proportional batch rank error calculated by $\epsilon_r = |v_e^r - v_v^r|/|\mathcal{P}^+|$ is used.

Figure 3.5 shows a moving average over 20 values of ϵ_r, $\forall r \in \mathcal{P}^+$, sorted by real batch rank (with 1 being the batch with the highest savings). For the best batch, the average is over batches ranked 1,..,20, for the second best, it is over 2,...,21 and so on. In SA-REMIX, the interest lies in batches with the highest savings, i.e., finding the best feasible combination of the best ranked batches. Figure 3.5 shows that the savings estimates for the best batches are much better than for the rest of the batches. This validates the performance of PCES as seen later in the results.

Comparison to Batching Heuristics from Literature The solution quality and the computation time of PCES are both compared to the following heuristics for maximum batch sizes three and four: C&W(i) and C&W(ii) from De Koster et al. (1999); the VNS-batching algorithm by Albareda-Sambola et al. (2009); ABHC by Henn and Wäscher (2012) which uses either C&W(i) (a new variant of the ABHC used in this chapter) or C&W(ii) (as in Henn and Wäscher (2012)) as the initial solution; and the generally best performing seed algorithm combination from De Koster et al. (1999) (seed selection by finding the order with the longest travel time, and order addition by the order that minimizes the sum of the distances from

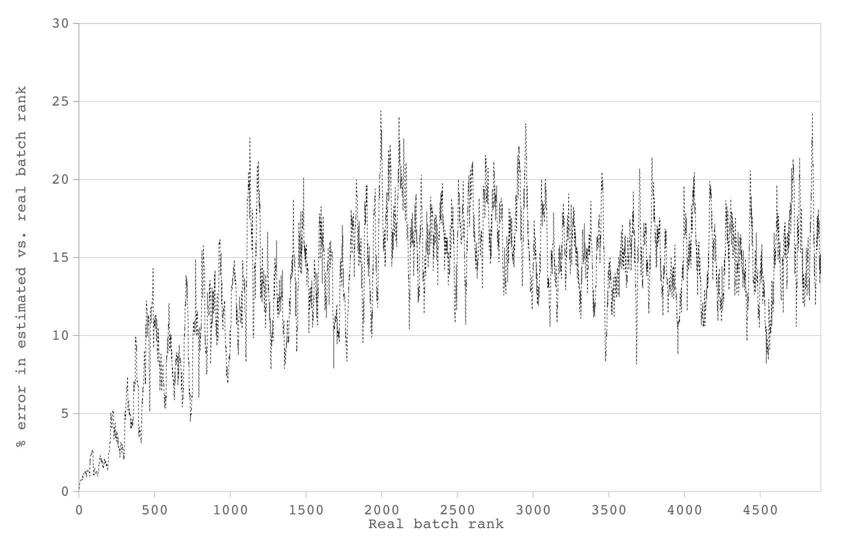


Figure 3.5. Error % in comparing batch ranks, ordered by the batch with the highest total rank for 10,000 batches of three customer orders sampled from a set of 1,536 customer orders. Only 4,900 are shown here so that small ranks can be distinguished.

the closest item in the seed to every item in the order). A 240 minute time limit is imposed for each wave size.

The results are shown in Figure 3.6. Figure 3.6a shows the percentage improvement compared to C&W(ii) in travel distance for wave sizes between 12-150, for all the other heuristics for a maximum batch size of three customer orders. C&W(i) is the weakest heuristic and always inferior to C&W(ii), as has been shown in the literature previously, see, e.g., De Koster et al. (1999) and Albareda-Sambola et al. (2009). However, C&W(i) is computationally light. The seed selection heuristic did not perform well, so it was left out of the results. Both the VNS and ABHC base their good results on detailed exploration of the combinatorial batch space, thus requiring a lot of batch evaluations. As the space gets larger with larger wave sizes, the number of batch evaluations increases exponentially. Initially, up to wave size 30, the VNS heuristic comes out on top — after this, the time limit for solving the problem kicks in, and the solution quality degrades rapidly, see Figure 3.6a. This is partly due to the fact that no initial solution is used - the search starts from each order being in a separate batch.

A similar effect imposed by the time limit and the computationally expensive batch evaluation can be seen with ABHC. Initially, it is the second best algorithm, but within the given time limit it can only make some improvements to the initial solution found with C&W(ii). Henn and Wäscher (2012) suggested that they got comparable results with initial solutions found with a FCFS-algorithm. However, during the tests it was found that using C&W(ii) to obtain a decent initial solution requires less computation time and produces better results. In general, PCES has better solution quality than C&W(ii) with less computation time (see Figure 3.6b). Also, for medium to large wavesizes, its solution quality is better than those of all the other methods, while simulataneously being much faster.

In Figure 3.7a the solution qualities of PCES and ABHC (initial solution with C&W(i)) are compared to C&W(i) for batches of four customer orders. The number of possible batch combinations is much larger than for batches of size three (see (6.9)). The routing is also computationally much more expensive. C&W(ii), VNS and thus also ABHC with C&W(ii) all fail to reach a proper solution result withing the time limit and are thus left out from the results. Using PCES yields a mean improvement of 2.3% compared to the batching of C&W(i), while ABHC is able to improve the C&W(i) initial solution by 0.01% on average, mostly because of the timelimit. Computational times of C&W(i) and PCES are similar for small wave sizes. PCES nears the timelimit as the number of iterations of SA-REMIX is increased quadratically with wave size, while ABHC always hits the time limit.

In both instances of three and four customer orders, PCES exhibits similar behaviour. For small wavesizes (up to 24) the error in the batch savings estimate dominates, and a good result is not reached reliably. However, recall that the estimate (Equations (3.16) and (3.18)) is quite good in estimating batches with high savings (see Figure 3.5), so as the number of such batches increases exponentially with both wave and batch size, the probability of finding a good solution increases with it.

Furthermore, it was found that the solution time is proportional to the number of batch evaluations of maximum batch size (three or four in this study).

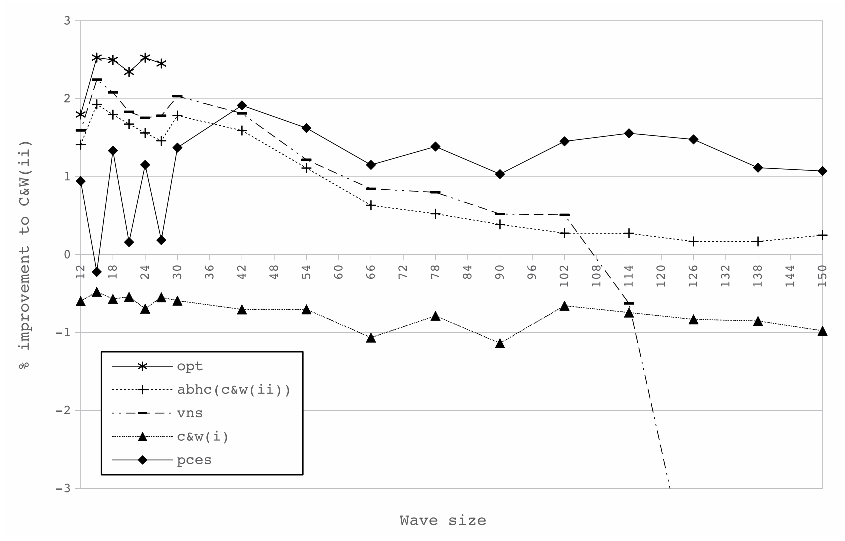
Comparison to optimal batching It is possible to compare the results of the batching procedure presented in this chapter, SA-REMIX, with optimal batching for up to 27 customer orders and batches of size three. The optimal routing algorithm is based on the A^*-routing method presented earlier in this chapter, while the order batching part uses dynamic programming, with all possible customer order combinations as state space.

For the optimal batching the wave size R is varied between 12 and 27, in steps of three customer orders. Figure 3.6a shows the results. For the optimal algorithm (*opt* in Figure 3.6a, see also Section 2.2.1 for the algorithm), the solution time increases exponentially with the wave size — calculating the optimal solution for the whole dataset of 1,536 orders with 27 customer orders per wave and $N = 3$ took over 23 hours. When compared to optimal solutions of waves of sizes between 12 and 27, PCES based batching gives results with less than a 1.2% error on average compared to the optimal solution for $N = 3$. Using PCES, when R increases, the time to solve the problem instance decreases. This is due to a reduction in calls to SA-REMIX. Note also the differences in computation time in Figure 3.6b.

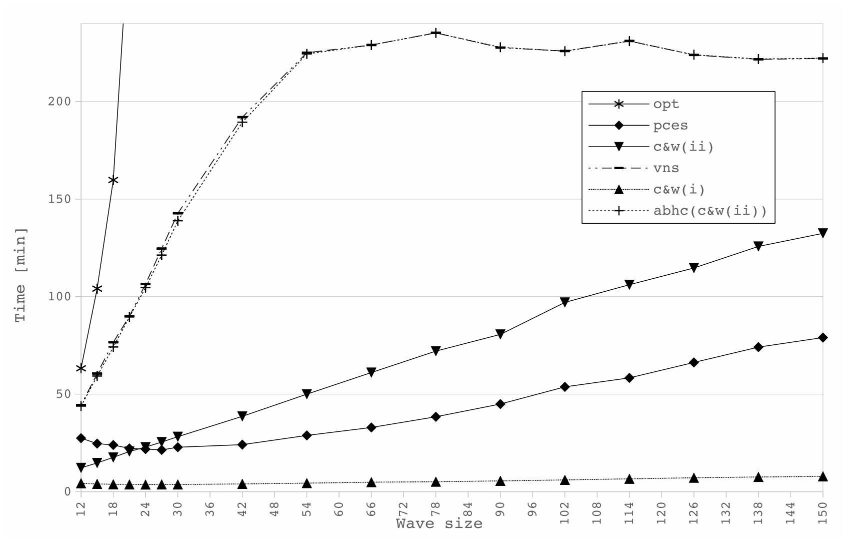
Effect of wave and batch size For batches of one customer order, wave size does not matter as each customer order is picked separately. For the batches of size two, travel distances are calculated with the package CPLEX 12.4 using the order-batching model presented by Gademann and Van de Velde (2005).

Figure 3.8 shows the benefits of different wave and batch sizes. Having more customer orders per wave reduces the total travel distance. For waves larger than 150 customer orders, there are no major changes in the different algoritms' relative performance. When comparing optimal batching for two customer orders to three customer orders, with a wave size of 27 customer orders, two customer order batches appear to lead to 19.4% longer travel distances. If the batch size is increased to four, there are still considerable savings to be had in travel distance when compared to those of batches of three customer orders. However, routing for these four-order instances is much more time consuming, and in practice this rules out the computation of an optimal baseline, as well as the application of most of the tested order batching heuristics from literature, apart from C&W(i).

Re-routing and rebatching of original batches Now the effect of PCES batching is compared with the company's current method, including the mandatory travel directions. Since the company's routing method is not optimal, an additional comparison is made to see how total travel distance is affected by just rerouting the original batches using the A*-routing method presented in Section 3.2.2. The results can be found in Table 3.1. By rerouting the original batches with a wave size of 126 customer orders, 3.1% or 980 km is saved, while rebatching and rerouting together

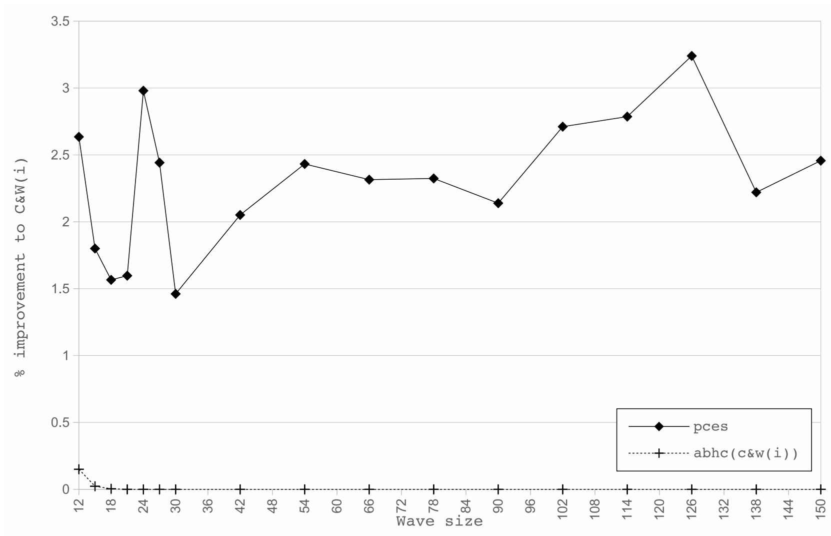


(a)

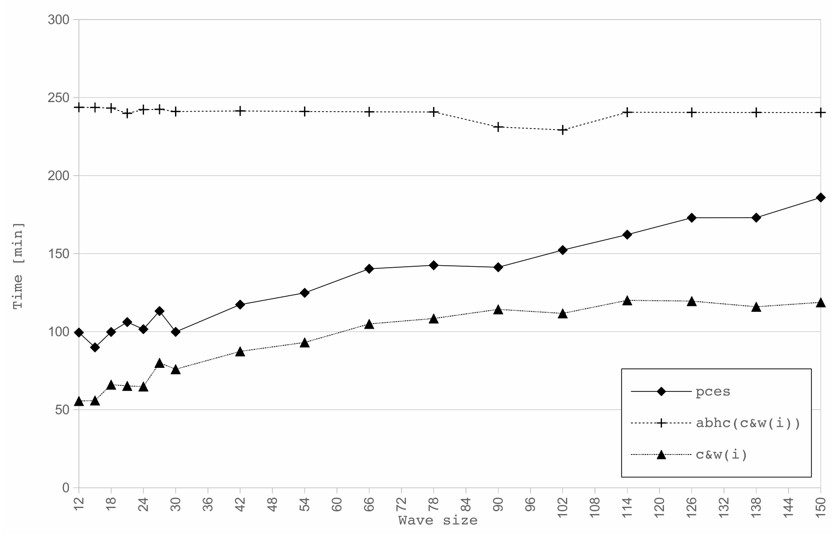


(b)

Figure 3.6. Comparison of PCES to other heuristics and an optimal solution for batch sizes of 3. The top figure shows the quality of the solution when compared to the C&W(ii), while the bottom one shows the computation time for each wave size for solving an instance of 1,536 orders.



(a)



(b)

Figure 3.7. Comparison of PCES to other heuristics and an optimal solution for batch sizes of 4. The top figure shows the quality of the solution when compared to the C&W(i), while the bottom one shows the computation time for each wave size for solving an instance of 1,536 orders.

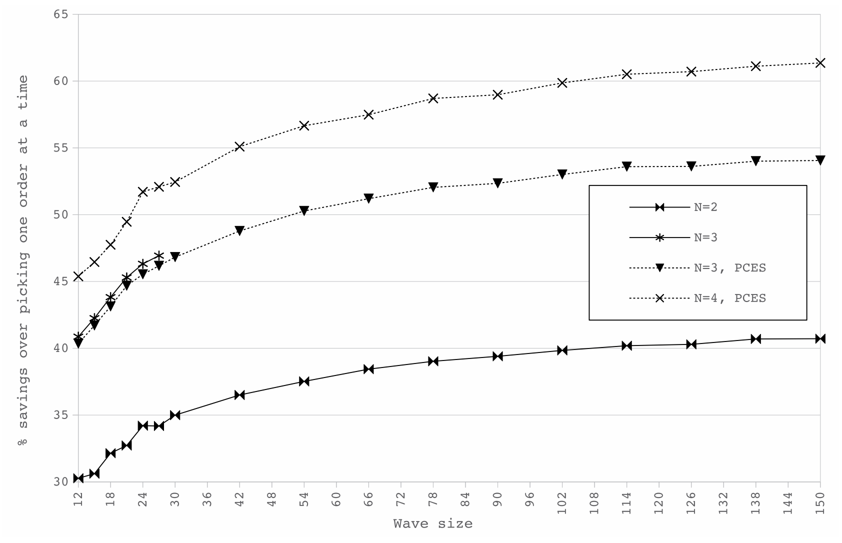


Figure 3.8. Percent of travel distance saved for optimal (solid lines) and PCES based batching (dashed lines) as a function of the wave size R in customer orders, for batches of size $N = 2$, $N = 3$, and $N = 4$, in comparison to $N = 1$. The mean error of PCES batching compared to optimal is 1.2% when $N = 3$.

Table 3.1. Results from rerouting and combined rerouting and rebatching of the original batches.

Month	ORG [km][1]	RR [km][2]	RRS [km][3]	RRS %[4]	RB [km][5]	RBS [km][6]	RBS %[7]
Jan	9698	9381	317	3.3%	8168	1530	15.8%
Feb	10189	9722	467	4.6%	8437	1752	17.2%
Mar	12060	11873	187	1.6%	10313	1748	14.5%
Total	31947	30967	980	3.1%	26918	5029	15.7%

[1] ORG is the travel distance resulting from the original batching and routing.
[2] RR is the rerouted travel distance with the original batching.
[3] RRS is the savings using the rerouted travel distance with the original batching.
[4] RRS % is the percentage of saved travel distance using the rerouted travel distances with the original batching.
[5] RB is the rerouted travel distance with PCES batching.
[6] RBS is the savings using the rerouted travel distance with PCES batching.
[7] RBS % is the percentage saved in travel distance with rerouting and PCES batching.

give savings of 15.7% or 5029 km.

3.4 Discussion

This chapter presents PCES batching based on optimal precedence-constrained routing in combination with a simulated annealing algorithm which uses estimated travel distance savings to group orders in batches. This chapter's contribution are in developing a generic method for solving the difficult problem of batching orders with strict sequence constraints. The method uses an estimate for savings generated by combining orders, which appears to perform remarkably well. The estimate uses exact savings from forming batches of size two to generate savings for larger batches. By using the estimate, the number of routings to be calculated reduces dramatically, particularly for larger batches.

The PCES algorithm is able to generate significant savings of up to 15.7% compared to the original batching, or over 5000 km for a three-month period. These savings can be achieved without having to make any changes to the warehouse layout or customer order composition. If the cost of order picking is 55% of total warehouse expenses (Drury, 1988) and 50 % (Tompkins et al., 2003) of this is due to travelling between picks, the savings on the total operating costs of the sample warehouse would be 4.3%.

The core idea behind PCES is to minimize the number of computationally complex batch evaluations. This makes it very competitive with other state-of-the-art batching heuristics, especially in solution time, but also in quality. Compared to other algorithms, which use a set of deterministic neighbourhood heuristics, the randomness inherent to the REMIX heuristic allows it to be flexible and perform well. For batches of three customer orders, PCES generally finds better solutions than the C&W(ii) algorithm, while its computational complexity is between C&W(i) and C&W(ii), and beating out other algorithms from literature for medium to large sized waves. When comparing performance to C&W(i), previous studies (Albareda-Sambola et al., 2009, Henn and Wäscher, 2012) have achieved proportionally higher savings than presented in this chapter. This is probably due to a larger maximum batch size and the fact that real orders are used in this study, whose composition is already somewhat optimized. However, as can be seen from the results in this chapter, PCES compares favorably to both the VNS and ABHC algorithms presented in Albareda-Sambola et al. (2009) and Henn and Wäscher (2012). For four order batches, PCES is peerless in solution quality with a 2.0% improvement over the next best algorithm. However, with small wave sizes of up to 30 customer orders, one would get better results using either of the VNS or ABHC heuristics, due to the inconsistent behaviour of PCES in that range (see Figure 3.6). This inconsistency is most likely due to fact that the probability for the estimate to be accurate is much smaller for small waves. As the number of good batches increases exponentially with wave size, so does the probability of the estimate being accurate.

When compared to optimal batching for small wave sizes of up to 27 customer orders and a maximum batch size of three, PCES performs decently, with a 1.2% mean error compared to the optimal solution. Savings in travel distance that are comparable to optimal batching and routing can be achieved with much less computation time. When compared to just picking one bin at a time, Figure 3.8 shows that increasing the batch size can result in substantial travel distance savings. Depending on the wave size, for batches of two orders, 30% to 40% is saved, while for batches of size three and four the savings range from 40% to 54% and 45% to 61%, respectively. Larger wave sizes seem to have a considerable effect on the total travel distance for up to about 100 customer orders. Wave sizes larger than this do not seem to affect solution quality very much.

Another benefit of the PCES batching method is that it allows us to evaluate the effect of different batch sizes in any warehouse with precedence-constrained routing. In case a different routing scheme is followed, it is rather straightforward to replace the routing algorithm presented in section 3.2.2 with this other method.

In practice, there may be further opportunities to reduce travel distance even more. New storage policies were not considered (see Dekker et al. (2004)), which might lead to additional savings. In addition, pickers' skills appear to vary significantly. Assigning the right pickers to the right batches (orders and products) will also impact total time. This is dealt with in Chapters 5 and 6.

4. Batching when Routing with a Generic TSP Algorithm

This chapter explores the performance of the batching part of PCES, the SA-REMIX algorithm, (see Chapter 3) when the precedence constrained routing is relaxed. The A* routing is replaced with a general-purpose TSP-solver, the Lin-Kernighan-Helsgaun algorithm (LKH) (Helsgaun, 2000). Theys et al. (2010) show that LKH is applicable to order picking problems irrespective of the warehouse layout, and that it provides good solutions (with a gap of 0.1%). SA-REMIX uses estimated savings for batches composed of three customer orders or more, as shown in Section 3.2.1. The results presented in Chapter 3 show that the algorithm performs very well when compared to other heuristics and optimal solutions. Central to this performance are the estimated savings, which are accurate for the most important part of the batches — those with high savings values (see Section 3.3). This chapter deals with the followin research question: how will SA-REMIX perform if the precedence constraints are relaxed and the A* algorithm is replaced with another routing algorithm? The results in this chapter show that SA-REMIX batching can be applied to any picker-to-parts order picking process in any warehouse not constrained by a particular layout or a specific routing method.

This chapter is organized as follows. First, the (non-precedence constrained) joint routing and batching problem is presented. Second, the experimental setup is detailed, followed by the results. The chapter concludes with a summary. LKH can be downloaded from Keld Helsgaun's website: `http://akira.ruc.dk/~keld/research/LKH/`.

4.1 Problem Description

In this section, the model presented in Section 3.1 is modified by eliminating the strict internal precedences. Furthermore, it is now assumed that no set drop-off locations exist, but all orders are dropped off at the depot (see Figure 3.1). This is done because LKH does not handle precedences. The model presented in Section 3.1 is modified as follows. Equation (3.7) is left out, and sets of first of and last items $\mathcal{G}_f$ and $\mathcal{G}_l$, used in (3.4) and (3.5), are not substracted from from the set of all items $\mathcal{G}$. $\mathcal{G}_f$ existed because each tour began by visiting one of the first items in each order, and this condition no longer applies due to the lack of precedence constraints. $\mathcal{G}_l$ is not used as it was composed of the drop-off locations, which are not used in this model. As with the previous model, each pick truck has a number of bins, to which orders can be assigned to. The objective is to minimize total travel distance.

Model Similarly as in Section 3.1, in order to model the batching and routing problem, all of the orders have to assigned to the batches. However, this time the sequence of items to be picked to each order is not relevant. To that end, let $\mathcal{P}$ be a set of potential batches. The number of potential batches should be sufficient to carry out all orders. Each batch can contain up to N bins. The set of bins is denoted by $\mathcal{B}$. The batch to which bin $b \in \mathcal{B}$ belongs is denoted by $r(b) \in \mathcal{P}$.

For each order $o \in \mathcal{O}$ and bin $b \in \mathcal{B}$ a binary decision variable $Y_{o,b}$ is introduced to indicate whether or not order o is picked in bin b. Furthermore, let $X_{g,k,r}$ be a binary decision variable indicating whether batch $r \in \mathcal{P}$ goes directly from item $g \in \mathcal{G}$ to item $k \in \mathcal{G}$. Finally, let $U_g \in \mathbb{N}$ indicate the relative position of item $g \in \mathcal{G}$ in the tour that picks the order of item g.

$$\min \sum_{g \in \mathcal{G}} \sum_{k \in \mathcal{G} \setminus \{g\}} d_{g,k} \sum_{r \in \mathcal{P}} X_{g,k,r} \tag{4.1}$$

subject to

$$\sum_{b \in \mathcal{B}} Y_{o,b} = 1 \qquad \forall o \in \mathcal{O} \tag{4.2}$$

$$\sum_{o \in \mathcal{O}} Y_{o,b} \leq 1 \qquad \forall b \in \mathcal{B} \tag{4.3}$$

$$\sum_{k \in \mathcal{G} \setminus \{g\}} X_{g,k,r} = \sum_{b \in \mathcal{B} : r(b) = r} Y_{o(g),b} \qquad \forall g \in \mathcal{G} \ \forall r \in \mathcal{P} \tag{4.4}$$

$$\sum_{g \in \mathcal{G} \setminus \{k\}} X_{g,k,r} = \sum_{b \in \mathcal{B} : r(b) = r} Y_{o(k),b} \qquad \forall k \in \mathcal{G} \ \forall r \in \mathcal{P} \tag{4.5}$$

$$U_g - U_k + |\mathcal{G}| \cdot \sum_{r \in R} X_{g,k,r} \leq |\mathcal{G}| - 1 \qquad \forall g, k \in \mathcal{G} : g \neq k \tag{4.6}$$

$$Y_{o,b} \in \{0, 1\} \qquad \forall o \in \mathcal{O} \ \forall b \in \mathcal{B} \tag{4.7}$$

$$X_{g,k,r} \in \{0, 1\} \qquad \forall g, k \in \mathcal{G} : g \neq k \ \forall r \in \mathcal{P} \tag{4.8}$$

$$U_g \in \mathbb{N} \qquad \forall g \in \mathcal{G} \tag{4.9}$$

Constraints (4.2) enforce the assignment of each order to exactly one bin, and (4.3) states that each bin covers at most one order. This results in a correct assignment of orders to bins.

Each location related to an item $g \in \mathcal{G}$ is exited so that the corresponding order is assigned to one of the bins in the batch as in (4.4). Equation (4.5) enforces that each item is entered so that the corresponding order is assigned to one of the bins in the batch. Thus, all items of an order are picked by the same batch. Constraints (4.6) are based on the subtour elimination constraints of (Miller et al., 1960). Constraints (4.7), (4.8) and (4.9) enforce non-negativity and binary qualities of the decision variables.

4.2 Results

Similarly as in Section 3.3, a set of 1536 orders is sampled from the data set of the Finnish retailer. The customer orders are grouped in waves, with the wave size R varying between 12 and 150 orders. The maximum batch size N is set to three. From (6.9), the total number of possible batch combinations per wave ranges from 298 (12 orders per wave) to 562625 (150 orders per wave). C&W(i) is used as the baseline algorithm for comparisons. It is compared to C&W(ii), SA-REMIX, the VNS by Albareda-

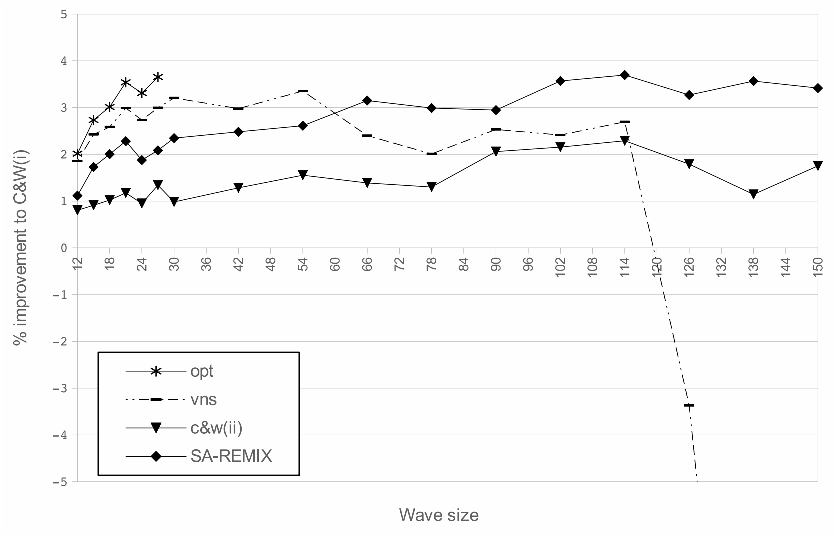
Sambola et al. (2009), and an optimal solution for small instances. C&W(i) is used as the initial solution for SA-REMIX. The number of iterations J used in the experiments is $J = 500,000 + R^2$, as the number of possible combinations increases with wavesize. For the algorithms to process all 1536 orders, a maximum timelimit of 10h is imposed.

Solution quality Figure 4.1a shows the solution quality of the algorithms. As in Section 3.3, VNS is the algorithm with the best solution quality for wave sizes smaller than 60. From that point on, SA-REMIX provides the best solutions, being on average 2.6% better than C&W(i). For waves equal or larger or than 60 orders is the next best algorithm is VNS (up to 114 orders, after which the timelimit kicks in), followed by C&W(ii). When compared to optimal batching and routing with LKH, the combination of LKH and SA-REMIX provides solutions that are on average 1.2% worse, which is in line with the results in Chapter 3.

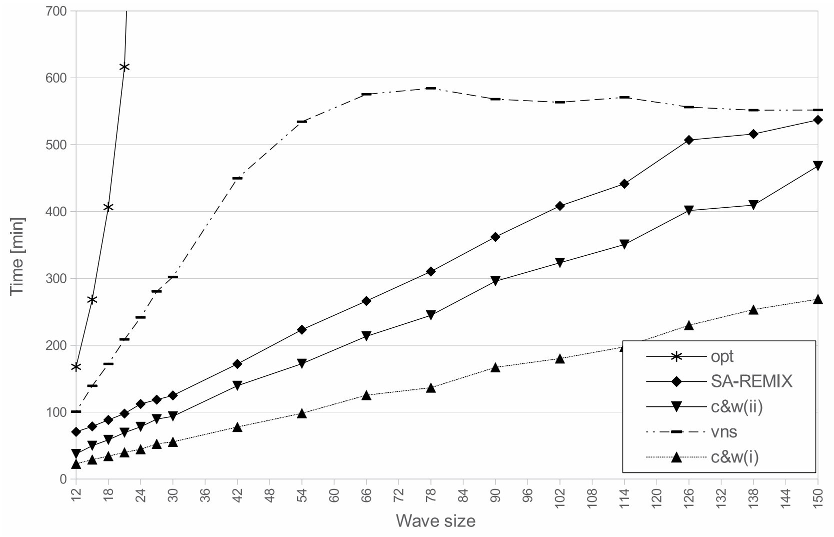
Computational time Due to the complexity of the algorithms that are used to solve joint picker routing and order batching problem, computational time was considered. Figure 4.1 shows the results for the tested algorithms by wave size. The C&W(i) and C&W(ii) algorithms are the faster ones, in that order, followed by SA-REMIX. This is a bit surprising, as previously in Chapter 3 the computational effort of SA-REMIX was seen to be somewhere in between the two C&W-algorithms. Undoubtedly the reason for this is the changing of the routing algorithm, at least indirectly. Moreover, the C++ implementation of the SA-REMIX algorithm can be further optimized, but this is left as future work. The VNS exhibits similar behavior as in Chapter 3, with steadily increasing computation time. Eventually finding a good solution to the batching problem with VNS becomes impossible due to the time constraint of 10h.

4.3 Discussion

The results show that using SA-REMIX, which is the batching part of the PCES algorithm, can also be used to solve batching problems without precedence constraints. Moreover, the algorithm provides good overall solution quality and the best performance within the comparison group with large wave sizes.



(a) Solution quality per wave size for different order batching algorithms compared to C&W(i).



(b) Computational time per wave size for different order batching algorithms.

Figure 4.1. Comparison of PCES to other heuristics and an optimal solution for batch sizes of 3. The top figure shows the quality of the solution when compared to the C&W(i), while the bottom one shows the computational time for each wave size.

5. Forecasting Batch Execution Time for Individual Pickers

Most warehouse management systems (WMS) store order picking logs, which are captured at a very detailed level by advanced picking tools. In this section, such log data are used to construct models to forecast the batch execution time for individual pickers. Batch execution time may depend on many factors, such as the details of the batch to be picked, but also on behavioral factors, such as intrinsic and extrinsic motivation and ability (Larco Martinelli, 2010). However, rather than explicitly including behavioral factors, they are implicitly included by only considering the past performance of each picker, as this is what can be found in the WMS data. WMS data of a picked order in a batch typically contain: picker ID, roll cage IDs in which the items are picked, drop-off locations of the roll containers, time-stamp of each *order line* (a part of customer order that has to be picked, also called an *item* or *pick line*), slot address per line, item IDs, number of units picked, etc. These data are used in a multi-level model, where the pickers form the "groups". Multilevel modeling is naturally suited for distinguishing between-group (i.e., between-picker) differences. It also scales well for groups of different amounts of input data. A group's random effects are allowed to deviate more from the fixed effects if it has a large amount of input data. As potential independent variables, the following are selected: number of pick lines in a batch, total batch travel distance, total pick item mass, and volume in a batch, as well as the mean pick height level at which items are picked during the picking tour.

First, the dataset is briefly described, followed by an explanation of the used regression method and the data cleaning. Finally, the regression results are shown. Cluster analysis is done on the picker models to illustrate the differences between the pickers.

5.1 Warehouse and Dataset Description

The sample warehouse in this chapter is a large picker-to-parts warehouse in Finland. Three months of extensive pick data from the warehouse's pick-by-voice system were obtained and processed. The warehouse has three cross-aisles, 57 aisles, seven drop-off locations, unidirectional travel in the aisles, and a single depot storing empty roll containers (see Figure 3.1 for the layout). Each customer order contains a maximum of 50 order lines, and is picked to a single roll container using a motorized truck that can transport a maximum of three roll containers. Thus batches consist of a maximum of three customer orders.

The data-set allows the extraction of original customer orders and batches and the identification of the picker who did the work. The batch execution time depends on: (1) picker ID, (2) total number of lines, (3) total travel distance calculated from the distance between all sequentially visited locations in a tour $[m]$, (4) total mass $[kg]$, (5) mean pick level, where 1 is low and 2 is high, and (6) total volume $[m^3]$.

The models are then used in an ALNS algorithm (as explained in sections 6.1, 6.3, and 6.4) to assign orders and batches to pickers. The whole dataset consists of 37,841 batches handled by 229 pickers during a three-month period.

5.2 Data Cleaning

To find possible outliers, the data were preprocessed. The results are summarized in Table 5.1.

Three different ways of picking were found: (i) multiple orders in a batch (pick tour), (ii) individual orders, and (iii) multiple orders with no apparent routing, and all lines with the same timestamp. The second and third methods can occur in exceptional situations and represent non-standard ways of working. The second occurs when picking heavy items to pallets instead of roll containers (*Pallet*), and the third occurs when the warehouse manager signs off previously picked orders by a single press of a button (*Timestamp*). The large majority of the batches (83%), represented by (i), is used in the regression modeling; (ii) and (iii) were not considered due to their representing a different way of working.

In some cases, trucks seemed to exceed the maximum speed of 10 km/h. Such batches are filtered out (*Speed*). Some items could not be mixed

Table 5.1. Summary of data cleaning. The cleaning categories are in the order they are applied — thus *Speed* is first and *Picker out* is last.

	Total	Pallet	Timestamp	Speed	Mass	Long time	Many pickers	Zeroes	Picker out	Remaining
Batches	37841	3257	3148	852	553	146	146	34	4070	24669
Percentage	100%	8.6 %	8.3 %	2.3%	0.1%	<0.1%	<0.1%	<0.1%	10.8%	65.2%

Table 5.2. Mean values and coefficients of variation per batch before and after data cleaning.

	Time $[min]$	Lines	Travel $[m]$	Mass $[kg]$	Level	Vol $[m^3]$
Before clean-up						
μ	29.6	46.3	598.5	206.5	1.1	1.0
CV	1.21	0.77	0.96	0.78	0.12	0.53
After clean-up						
μ	31.7	51.8	620.3	224.0	1.1	1.0
CV	0.69	0.65	0.63	0.67	0.11	0.40

with other products, e.g., because they were highly fragile. The weight of such line items is artificially inflated to reach the threshold weight of 600 kg per order. Orders containing such lines were discarded (*Mass*). In addition, batches that took too long to execute were also discarded. This can be caused by breaks, shift changes in the picking process, or task interruptions with other causes. Based on experience in warehouses with similar picking processes, the threshold was set at a maximum of 2 hours net picking time per batch (*Long time*). Batches that were picked by more than one picker were also excluded (*Many pickers*).

Neither the output nor the inputs were allowed to contain zero values — all such batches were omitted from consideration (*Zeroes*). Finally, to construct a reasonably large set of cross-validation data, it is required for each picker included in the model to have at least 75 batches of input data (80% of which are used for tuning the model) (*Picker out*).

After the data cleaning and the requirement on the number of data lines, 99 pickers out of 229 qualified. Table 5.1 shows a summary of the results of the data cleaning. After completing the data cleaning, 24,669 batches remained to be used for the modeling of the pickers.

Table 5.2 lists the means and coefficients of variation (*CV*) for the inputs and outputs. Most of the CV values are smaller than one: this suggests that many of the tasks (batches) that the pickers have done are quite similar, or at least comparable. In the non-cleaned data, batch execution time can vary quite a bit, most probably due to breaks, etc., that are filtered out in the data cleaning. Batch parameters *Lines*, *Travel* and *Mass* seem to vary the most, while *Level* seems to vary very little: most of the picking is done from the low level.

5.3 Multilevel Modeling

Let $w \in \mathcal{W}$ denote the groups (the pickers) and $r \in \mathcal{R}$ are indices to the data (the batches). Furthermore, let vector $\boldsymbol{\beta}_w$ contain the model parameters for group w and let $t_{r,w}$ be the output for a forecasting model, which forecasts total batch processing time, for some r and w. Using the notation of Bryk and Raudenbush (1992), a linear multilevel forecasting model can be formulated as in equations (5.1).

$$t_{r,w} = \beta_{0,w} + \sum_{i=1}^{n} \beta_{i,w} x_{i,r} + \kappa_{w,r} \tag{5.1a}$$

$$\beta_{i,w} = \gamma_i + u_{i,w} \quad \forall i \in \{0, ..., n\}, \tag{5.1b}$$

where $x_{i,r}$ is the i:th element of input vector $\boldsymbol{x}_r$, $\kappa_{w,r}$ is the within group error term, and γ_i is the slope effect of the dependent variable $i \in \{1, ..., n\}$. For group w, $u_{0,w}$ is the intercept error and $u_{i,w}$ are slope errors $\forall i \in \{1, ..., n\}$.

A two-level multilevel model is used to forecast each picker's pick time per batch. Following Bliese (2002), three important sources of variation are found: within group (σ^2), between-group variation in intercepts (τ_0), and between-group variation in slopes (τ_1). The models are built using R software and its packages, *multilevel* and *nlme*.

The data is divided in two parts. For each picker, a random 80% of the batches are used for regression, and the rest (20%) for cross-validation purposes only. By making this division, values for the cross-validation residuals can be found for all batches used in the validation. These residuals give us an idea of the accuracy of the model along with other measures, such as the Akaike Information Criterion (AIC) and applicable versions of R^2. As the forecasting models are directly used to calculate estimates of batch execution times in Section 6.4, the forecasts should be as accurate as possible. In regressing, the batches are ordered by timestamp to accommodate possible learning-based autocorrelations in the data.

Plots of batch execution times and independent variables show that variance increases with total batch execution time, i.e., there is heteroscedasticity in the data, which appears to be largely remedied by applying a logarithmic transform of the output data (batch execution times).

First a level-1 model is constructed, i.e., a model with between-pickers variability only in the intercept. Second, the level-1 model is extended to level-2 by allowing between-picker variability in the slopes.

Step 1: intercept variation. In this section, a test is made for significance of between-picker variation in the intercept. If the variation is not significant, it will not matter which picker gets a job — everyone will perform in a similar manner. Hence, solving any assignment problem will be pointless. However, if there are between-pickers differences, one should strive for a better assignment of work.

Independents are added to the model stepwise and tested whether there is significant variation between pickers. The variability of the intercept term is examined with a level-1 *Null Model* (Bryk and Raudenbush, 1992):

$$t_{r,w} = e^{(\beta_{0,w}+\kappa_{w,r})} E(e^{\kappa_{w,r}}) \tag{5.2a}$$

$$\beta_{0,w} = \gamma_0 + u_{0,w}. \tag{5.2b}$$

where $t_{r,w}$ is batch execution time of batch r for picker (group) w, γ_0 is the common intercept, $\kappa_{w,r}$ the within-group error term and $u_{0,w}$ the between-group error term, and $E(e^{\kappa_{w,r}})$ is the Smearing Estimate (Duan, 1983) of picker w (i.e., the expected value of the retransformed residuals) used to correct the model bias resulting from the non-linearity of retransforming the logarithmic dependent $t_{r,w}$. In combined form, the model is $t_{r,w} = e^{(\gamma_0+u_{0,w}+\kappa_{w,r})} E(e^{\kappa_{w,r}})$.

The Null Model has two possible sources of variance, τ_0 for how much the groups' intercept varies from the overall intercept (γ_0), and σ^2 for the within-group variance. The Intraclass Correlation Coefficient $ICC(1) = \tau_0/(\tau_0 + \sigma^2)$ (Bryk and Raudenbush, 1992, Kreft and De Leeuw, 1998) equals 0.103, implying 10.3% of the total variance in the natural logarithm of time is due or related to differences between pickers. This suggests that it is beneficial to assign the right batches to the right pickers.

Analysis of variance (ANOVA) is used to test for the significant difference in -2 log-likelihood ratios between a model with and a model without a random error term in the intercept (Bliese, 2002). A model with a high -2 log-likelihood is better than one with a low one, and the statistical significance of the difference is tested. To achieve this, a comparison is made between the Null Model and a generalized least squares (GLS) fit of a similar intercept-only model which does not contain the between-group error term. The null hypothesis that no significant difference in the -2 log-likelihood scores exists is rejected ($p < 0.0001$), so individual aspects explain up to 10.3% of the total variance in log-transformed batch execution time.

A Null Model can be built for non-transformed time, similarly as above. In this case $ICC(1) = 0.131$, i.e., 13.1% of the total variance in time is due to differences among pickers. The null hypothesis is rejected as well, so the result is significant. Such differences among pickers can arise from different physical or mental characteristics of the pickers, such as strength or height, motivation, experience or stacking ability.

Step 2: slope variation and model selection. The level 1 multilevel model is now extended into a level-2 model by including group level errors for slope. For n independent variables the model can be formulated as in equations (5.3) with the similar notation as in (5.1).

$$t_{r,w} = e^{(\beta_{0,w}+\sum_{i=1}^{n}\beta_{i,w}x_{i,r}+\kappa_{w,r})}E(e^{\kappa_{w,r}}) \tag{5.3a}$$

$$\beta_{i,w} = \gamma_i + u_{i,w} \quad \forall i \in \{0,...,n\} \tag{5.3b}$$

A model is selected using a stepwise procedure. The selection procedure is summarized in Table 5.3.

Each iteration, a new independent term is added to the current model. ANOVA is used to test whether a random effect should be added to the new model. This is done by comparing the -2 log-likelihoods of two versions of the new model: the first with only a fixed effect for the new term, the second with both fixed and random effects. Models are selected based first on the AIC and second on the sum of squared-error residuals (SS) of the set of batches used for cross-validation.

The procedure begins by adding the first term, number of lines (*Lines*) to the Null Model (see Table 5.3). To better fit the data to a normal distribution, two models are compared: one with log-transformed input data (Model 1L) and one with non-transformed data (Model 1X). Comparing AIC and SS for both models shows that Model 1L is better. A random effect cannot be excluded ($p < 0.0001$). Next, the second independent variable (*Travel*) is added to Model 1L, resulting in models 2LX (non-transformed variable) and 2LL (transformed variable). Both models are also tested for inclusion of random effects. The transformed input is better in terms AIC and SS for Model 2LL than for Model 2LX; therefore Model 2LL is chosen. The process continues by adding each of the

Table 5.3. Multilevel model selection procedure

Model[1]	Int(0)	Lines(1)	Travel(2)	Mass(3)	Level(4)	Volume(5)	AIC[4]	SS [5]	R^2_m [6]	R^2_c [6]	p(rand)[7]
0	***[2]	NA[3]	NA	NA	NA	NA	46006	5.6719e+09	0	0.10	<0.0001
1X	***	***	NA	NA	NA	NA	26631	4.6986e+09	0.59	0.68	<0.0001
1L	***	***	NA	NA	NA	NA	16834	2.2053e+09	0.68	0.82	<0.0001
2LX	***	***	***	NA	NA	NA	13922	2.0428e+09	0.71	0.84	0.0498
2LL	***	***	***	NA	NA	NA	13225	1.8963e+09	0.64	0.86	<0.0001
3LLX	***	***	***	***	NA	NA	12675	1.9143e+09	0.65	0.85	<0.0001
3LLL	***	***	***	***	NA	NA	12345	1.8727e+09	0.61	0.87	<0.0001
4LLLX	***	***	***	***	***	NA	11996	1.8517e+09	0.55	0.89	<0.0001
4LLLL	***	***	***	***	***	NA	11992	1.8510e+09	0.66	0.87	<0.0001
5LLLLX	***	***	***	***	***	***	11688	1.8290e+09	0.66	0.87	<0.0001
5LLLLL[8]											
OLS/GLS[9]	***	***	***	***	***	***	17780	2.5847e+09	0.79	-	-

[1] The model number. Zero indicates the intercept-only model. The characters following the model number indicate the modification done to a previous model. If an independent is added, the notation is: not included (N); in a non-transformed form with a random effect (X); or log-transformed form with a random effect (L). The addition of an autocorrelation model is denoted by (A). The last addition is the rightmost character.
[2] Confidence in the fixed-effect independent, "***" indicates $p \leq 0.001$, "-" that the independent is not significant.
[3] Independent not included in the model.
[4] The Akaike information criterion of the model, a measure of information lost if the model is used to represent the data. Lower is better.
[5] Sum of squared residuals resulting from the cross-validation of the models.
[6] R^2_m is used to describe the proportion of variance explained only by fixed factors, while R^2_c is used to describe the variance explained by both fixed and random (between-picker) factors (Nakagawa and Schielzeth, 2013).
[7] ANOVA test result on whether a between-pickers error term (random effect) should not be included in the model.
[8] The regression did not converge for this model.
[9] An OLS/GLS multivariate regression over the whole data, with some of the variables log-transformed as with the best model 5LLLLX (the coefficients of both the OLS and GLS models are the same).

log-transformed independent variables and the non-transformed ones, always testing for the significance of the random effect. The tested models are shown after Model 1L in Table 5.3. This results in Model 5LLLLX, which has logarithmic inputs for all other terms apart from *Volume*.

The addition of interaction terms between the independent variables to Model 5LLLLX did not significantly improve any fit or error measure, or the regression did not converge. The final model in the notation of Bryk and Raudenbush (1992) is

$$\begin{aligned} a(\boldsymbol{\beta}_w, \boldsymbol{x}_r) =& \beta_{0,w} + \beta_{1,w}\ln(x_{1,r}) + \beta_{2,w}\ln(x_{2,r}) + \beta_{3,w}\ln(x_{3,r}) \\ & + \beta_{4,w}\ln(x_{4,r}) + \beta_{5,w}x_{5,r} + \kappa_{w,r} \qquad \text{(5.4a)} \\ t_{r,w} =& e^{a(\boldsymbol{\beta}_w, \boldsymbol{x}_r)} E(e^{\kappa_{w,r}}) \qquad \text{(5.4b)} \\ \beta_{i,w} =& \gamma_i + u_{i,w} \quad \forall i \in \{0, ..., 5\} \qquad \text{(5.4c)} \end{aligned}$$

To test for multicollinearity, a single multiple linear regression model was built with the variables of Model 5LLLLX and run over all data. The maximum VIF value was 4.4 (for $\ln(Lines)$) suggesting that multicollinearity is not a major issue.

Figure 5.1 shows the cross-validation residuals grouped per picker. The pickers' residuals appear to be zero mean, and for approximately half of the pickers, the models seem to relatively accurate. However, there is still clear heteroscedasticity in approximately half of the pickers — batch time residuals are mainly skewed to the right, as is the case with the original data. However, the consistent reduction of the sum of squares of cross-validation residuals SS as the model selection progresses in Table 5.3 is encouraging, despite the apparent heteroscedasticity.

Multivariate OLS and GLS models are fitted using all independent variables, some log-transformed as in Model 5LLLLX, over the whole data set. Between-picker differences are not taken into account. The model coefficients in the fitted OLS and GLS models coincide. ANOVA is used to test the null hypothesis of Model 5LLLLX not having a significantly better -2 log-likelihood value compared to the GLS model. The null hypothesis is rejected with $p < 0.0001$.

The resulting elasticities and standard deviations of model 5LLLLX and the corresponding random effects can be found in Table 5.4. The betas (elasticities) show that the number of lines in a batch is the most important factor in the model. The second row shows the between-picker

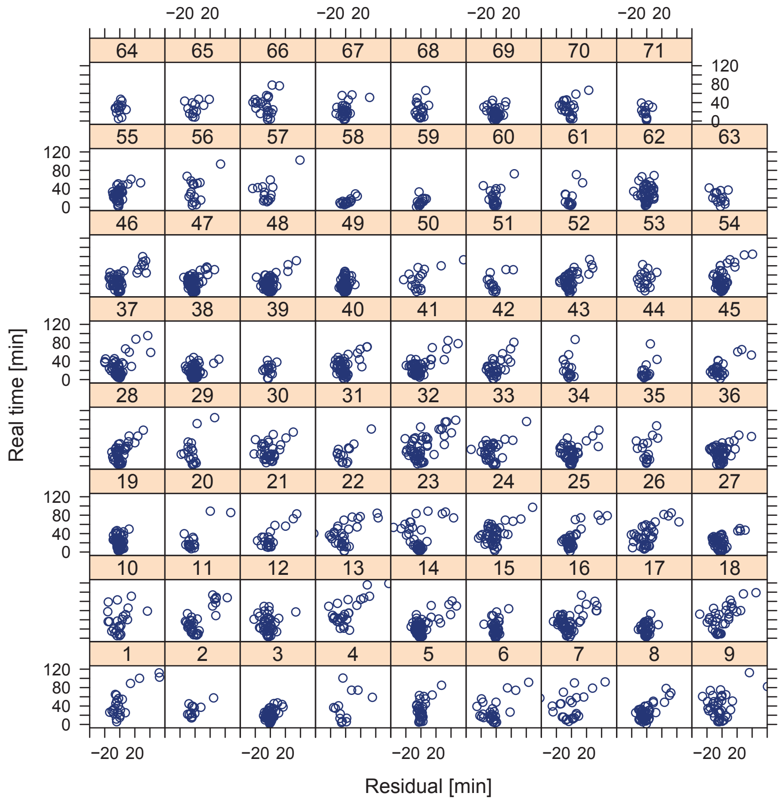


Figure 5.1. Cross-validation residuals for final full model vs. batch time, grouped per picker.

standard deviation of the model. The standard deviation of $ln(Lines)$, $ln(Travel)$ and $ln(Mass)$ are quite similar. The effect of mean picking level seems to vary the most among the picker models.

The effect of increasing any of the log-transformed inputs is proportional to the (re-transformed) output in this model. The dependent variable is log-transformed, as are most of the independent variables, apart from *Volume*. In the case, when both are log-transformed, the scaled model beta can directly be used to calculate the effect of an independent variable on the dependent variable. For example, for the fixed effect model, an 1% increase in the number of lines in a batch (*Lines*) results in an 0.66% increase in total batch execution time. For the non-transformed independent variable, *Volume*, a unit increase ($1m^3$) will reduce the batch execution time by 0.16%. For Model 5LLLLX, the standard deviation of the residual is 0.32, which means that if the forecasted time for a picker to pick a batch is $t_{w,r}$, there is an approximate 68% chance that it is accurate by a factor of $e^{0.32} = 1.37$, i.e, it is in the range $[t_{w,r}/1.37 \quad 1.37t_{w,r}]$

Table 5.4. Model betas for 5LLLLX with scaled data, effects on $ln(Time)$.

	Intercept	ln(Lines)	ln(Travel)	ln(Mass)	ln(Level)	Vol
β	3.2	0.66	0.16	0.17	-0.44	-0.16
Random effect stdev	0.40	0.09	0.04	0.08	0.38	0.12

(Gelman and Hill, 2007, p. 62). Variance of batch execution time thus increases proportionally to the value of time.

Table 5.4 shows that the batch execution time of the average picker grows with the number of order lines, the distance traveled and total batch mass, all of which seem reasonable: as the amount of work, travel or mass increases, it is natural that time increases as well. A higher pick level seems to make picking quicker, which suggests that currently the pickers might be mostly picking from a non-optimal height. The negative coefficient in volume is somewhat more difficult to explain. This might result from the effect that some of the batches that contain relatively few items of high volume are fast to stack. In some sense, the volume coefficient can also reflect the stacking skill of the picker.

Picker clusters Each of the pickers has their own forecasting model. Generalizing a bit, and to explain the coefficients in terms of individual skill, one could characterize the pickers based on on the model coefficients in the following way:

- a low $ln(Lines)$ represents a generally good worker;
- a picker with a low $ln(Travel)$ is most likely a quick driver;
- a low $ln(Mass)$ represents strenght;
- a low $ln(Level)$ most likely means that the picker is tall, or has an easier time picking items from the high level;
- a low Vol represents a picker with good stacking ability.

To group and illustrate similarities among pickers, hierarchical clustering is performed on the forecasting models using R software and the function *hclust*. Euclidian distance between the picker coefficients is used as the metric, and chosen mainly for its simplicity. The hierarchical clustering routine is agglomerative, i.e., it first assigns a cluster for each of the pickers and then iteratively proceeds to merge the most similar clusters. In

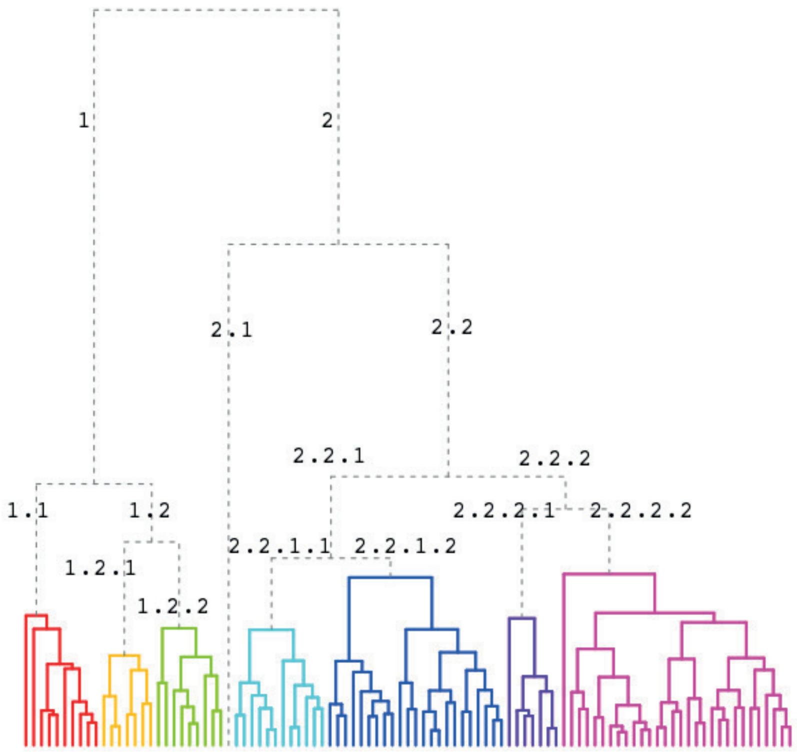


Figure 5.2. Dendogram of picker groups formed with hierachical clustering

Figure 5.2 the results are illustrated with a dendogram. Picker clusters are numbered by size: 1 and 2 indicate the two largest clusters. If a cluster is divided, a minor number is added to the current clusters id, e.g., cluster 1 is split into two subclusters indicated by 1.1 and 1.2 in Figure 5.2.

Table 5.5. Picker clusters, cluster sizes and mean coefficients ordered by cluster height in Figure 5.2.

Cluster id	size	ln(Lines)	ln(Travel)	ln(Mass)	ln(Level)	Vol
1	26	0.69	0.17	0.18	-0.04	-0.15
2	73	0.65	0.16	0.17	-0.59	-0.16
2.1	1	0.70	0.15	0.15	-1.42	-0.48
2.2	72	0.65	0.16	0.17	-0.57	-0.15
2.2.1	35	0.65	0.16	0.17	-0.71	-0.17
2.2.2	37	0.65	0.16	0.18	-0.45	-0.14
1.1	10	0.72	0.16	0.17	-0.20	-0.22
1.2	16	0.67	0.18	0.19	0.05	-0.10
2.2.2.1	7	0.75	0.17	0.09	-0.52	-0.11
2.2.2.2	30	0.63	0.16	0.20	-0.43	-0.14
1.2.1	7	0.60	0.17	0.22	-0.05	-0.05
1.2.2	9	0.73	0.18	0.17	-0.13	-0.14
2.2.1.1	12	0.71	0.14	0.17	-0.66	-0.26
2.2.1.2	23	0.62	0.17	0.17	-0.74	-0.13
Overall	99	0.66	0.16	0.17	-0.44	-0.16

Table 5.5 shows the mean picker coefficients for each numbered cluster in Figure 5.2 and the number of pickers present in each cluster. The pickers in cluster 2 are in general faster than those in cluster 1. In particular, this can be seen from the differences in by the coefficients for $ln(Lines)$ and $ln(Level)$. It could be said, that the pickers in cluster 2 are more adept in picking batches with larger numbers of order lines and benefit from more from the items being on a higher level. Continuing to follow the dendogram from top to bottom, it can be observed that the next cluster split occurs to cluster 2: a single picker deviates from this cluster to form 2.1. He/she seems adept in picking items from a high elevation, and batches with relatively high volume. Cluster 2.2 contains the rest of cluster 2, and is further split to 2.2.1 and 2.2.2, where the pickers in 2.2.1 seem to benefit more from picking from a higher elevation when compared to 2.2.2. Next, cluster 1, which contains (at least some of) the slow pickers, splits to 1.1 and 1.2: the pickers in 1.1 are slower in picking batches with many order lines but the pickers in 1.2 are slower to travel and adversely affected by pick level, where they have the only positive coefficient in $ln(Level)$ among all the clusters. Cluster 1.2 also benefits less from batches with high volume. Clusters 2.2.2.1 and 2.2.2.2 are separated next. The pickers in 2.2.2.1 are less affected by batches with high mass than any of the other groups, but are not particularly fast in picking batches with many order lines, while opposite is true for 2.2.2.2. Cluster 1.2.1 contains pickers that are good with batches with many order lines but are affected heavily by batch mass and benefit less from pick level and batch volume. Cluster 1.2.2 contains pickers who are slow in picking most batches with many lines, and don't benefit much from the pick level or batch volume. Finally, clusters 2.2.1.1 and 2.2.1.2 contain the pickers

that benefit most from the high pick level but are either slow with batches with many orders and fast to travel (2.2.1.1) or are fast with many order batches (2.2.1.2).

From the analysis above, it can be seen that differences among the picker clusters exist in all factors. The pickers have different strengths and weaknesses that become apparent when comparing to the mean picker (coefficients).

5.4 Discussion

In this chapter, it has been shown that a significant part, i.e., 13.1%, of the total batch execution time is due to differences in pickers. This implies that it can be possible to find the right picker for each batch. The coefficients of variation of the input data are all less than one suggesting that batches are in general quite similar. A higher variance in the batch parameters in the data (see Table 5.2) would most likely lead to a worse fit. This would result in less accurate picker models, and consequently to more inaccurate forecasts. However, analysis of variance tests should in this case indicate that something is amiss. Inaccuracies in the forecasting models would carry over to the optimization presented in the next chapter. A multilevel model is fit to the cleaned-up data, using log-transformed data to counter heteroscedasticity. The model is systematically built using a stepwise procedure based on improvements in both cross-validation criteria and AIC. As AIC decreases, the cross-validation criteria generally does as well. This confirms the sanity of the model selection procedure. All possible inputs (or their transformed versions) are shown to be statistically significant. Furthermore, when comparing the sum-of-squares cross-validation residuals between the OLS/GLS and final multilevel models, it is apparent that the multilevel model is superior. This reinforces the hypothesis that differences among pickers are significant. Generally, transformed variables provide a better fit than non-transformed ones. This is perhaps partly due to increased error probability in task execution as the complexity of the task increases. In practice, the forecasting models should be periodically updated to account for possible learning of pickers.

6. Assigning Orders and Batches to Pickers of Varying Skill

Data on the picking process and individual order pickers, readily available in many warehouses, can be used to significantly improve the efficiency of the order-picking process. This is illustrated for the case of a Finnish retailer operating a single large order picking warehouse supplying all of its stores. By optimizing travel distance of order pickers, some savings in total batch execution time can be gained. Instead of considering just travel time, if factors such as setup, search, and picking time are also included, process efficiency can be significantly improved. The results of Chapter 5 show that significant differences exist among pickers. In this light, and as no previous work has included picker skills in optimizing order picking operations, the forecasting models built in Chapter 5 are used to determine which orders should be combined in a pick tour (i.e., a batch) and formulate the problem as a joint order batching and generalized assignment (BatchGAP) model. In this model, batches are assigned to pickers with the objective of minimizing the sum of total batch execution times. Comparisons to the current state-of-the-art in order picking show that over 9% of total batch execution time can be saved by considering picker skill and work assignment. These results can be used by warehouse managers to either dispatch the right order to the best from the available pickers, or in the case of hiring *flex pickers*, to consult past log data and hire those pickers matching the current orders.

6.1 Joint Batching and Generalized Assignment Problem

As the objective of the optimization presented in this section, customer orders and batches are assigned to those order pickers who have the best skills to execute them, thus minimizing the total batch execution time (including pick, travel, and setup time).

Order batching for batch sizes of three or more orders has been shown to be NP-hard (Gademann and Van de Velde, 2005). To further assign batches to pickers, the order batching model from Gademann and Van de Velde (2005) is extended to include a generalized assignment problem, which is also NP-hard (Fisher et al., 1986). This results in a joint batching and generalized assignment problem (BatchGAP), which is sufficiently complex to justify the use of a heuristic.

The problem is defined as follows. Let $\mathcal{R}$ be the set of all possible batch combinations from the set of orders $\mathcal{O}$, each batch consisting of a maximum of N orders. The orders that are to be picked during a day arrive early each morning. Let $\mathcal{R}_s \subseteq \mathcal{R}$ be the set of batches present in solution $s \in \mathcal{S}$, where $\mathcal{S}$ is the set of feasible and complete solutions to the BatchGAP. Furthermore, let $\mathcal{W}$ be the set of modeled pickers. The mapping of each customer order in $\mathcal{O}$ to a batch $r \in \mathcal{R}$ is characterized with a zero-one vector $\boldsymbol{a_r}$ of length $|\mathcal{O}|$. If $a_{or} = 1$, order o is included in batch r, otherwise it is not. Each batch $r \in \mathcal{R}$ has multiple order lines that need to be picked in no particular order during a picking tour $y \in \mathcal{Y}_r$, where $\mathcal{Y}_r$ is the set of all possible tours for batch r.

After picking all lines of an order, the order must be left at an order-specific drop-off location in the warehouse. All pick and drop-off locations must be visited at least once. Thus a tour is a solution to a TSP, where the drop-off locations have to be visited after the last pick in the batch (see Figure 3.1). All picking tours start and end at a depot. Each batch $r \in \mathcal{R}$ has a travel distance of d_r associated with the tour-construction method used. The forecast total batch execution time, $t_{w,r}$, depends on the parameters of batch r and the forecasting model of picker w. When calculating the forecast $t_{w,r}$, a TSP corresponding to the batch r to get the travel distance d_r needs to be solved. The TSP is solved separately with a heuristic (see Section 6.3.1). The TSP constraints are not included in the model below, as the model is mainly illustrative of the problem, and the additional constraints would unnecessarily complicate it. A heuristic is used to solve the TSP as no fast optimal solution to a precedence-constrained TSP (note that the last location of each order is fixed) exists for a warehouse with a middle cross-aisle and unidirectional travel. The tour length of a TSP related to a particular batch is used as one of the inputs when forecasting $t_{w,r}$ using equations (5.4). Each picker $w \in \mathcal{W}$ has a maximum working time during a shift, M_w.

The following optimization model can be formulated for the BatchGAP:

$$\min \sum_{w \in \mathcal{W}} \sum_{r \in \mathcal{R}} t_{w,r} X_{w,r} \tag{6.1}$$

subject to

$$\sum_{w \in \mathcal{W}} X_{w,r} \leq 1 \quad \forall r \in \mathcal{R} \tag{6.2}$$

$$\sum_{r \in \mathcal{R}} t_{w,r} X_{w,r} \leq M_w \quad \forall w \in \mathcal{W} \tag{6.3}$$

$$\sum_{o \in \mathcal{O}} a_{o,r} \leq N \quad \forall r \in \mathcal{R} \tag{6.4}$$

$$\sum_{w \in \mathcal{W}} \sum_{r \in \mathcal{R}} a_{o,r} X_{w,r} = 1 \quad \forall o \in \mathcal{O} \tag{6.5}$$

$$X_{w,r} \in \{0,1\} \tag{6.6}$$

The goal is to find a solution s^* such that the set of batches $\mathcal{R}_{s*} \subseteq \mathcal{R}$ form a complete partitioning of $\mathcal{O}$ and have a feasible assignment to the pickers $\mathcal{W}$, such that the sum of costs of all pickers is minimized and constraints (6.2, 6.3, 6.4 and 6.5) are upheld.

The objective function (6.1) minimizes the sum of all the pickers' batch execution times. The binary decision variable $X_{w,r}$ is one if batch r is assigned to picker w, otherwise it is zero. Constraints (6.2) enforce that each batch is picked at most once. Each pickers' maximum working time, M_w is enforced by (6.3). Constraints (6.4) enforce that no more than N orders are contained in any one batch. Each order must be allocated once to any chosen batch (6.5) and integrality of the decision variable $X_{w,r}$ is enforced by (6.6).

As a by-product of the optimization, the effect on picker productivity is also of interest. Let l_r be the number of order lines in batch r. Productivity of a picker $w \in \mathcal{W}$ is defined as

$$L_w = \sum_{r \in \mathcal{R}} X_{w,r} l_r / \sum_{r \in \mathcal{R}} X_{w,r} t_{w,r} \tag{6.7}$$

or the average number of order lines picked per unit of time worked (usually minutes). Notice that the total work time differs from maximum shift length. This study aims to model the maximum the maximum total time that can be saved — salary models are not considered.

6.2 Note on Problem Complexity

The complexity of assigning orders to batches and batches to pickers can be illustrated by the number of combinations of a related bin-packing problem, where batches and pickers are both homogeneous. The number of possible batch combinations of up to the maximum batch size of N can be found from (6.9). Let $|\mathcal{P}|$ be the total number of batches, and let L be the number of batches a picker can complete during a shift. Assuming $|\mathcal{P}| = |\mathcal{W}|L$, the total number of possible bin-packing combinations for $|\mathcal{W}|$ pickers is

$$C_{assign} = \sum_{i=1}^{|\mathcal{W}|} \begin{pmatrix} |\mathcal{P}| - iL \\ L \end{pmatrix}. \tag{6.8}$$

A regular shift at the sample warehouse deploys by 30 pickers. Let one picker take half an hour to process one batch, and work 8h each shift, for a maximum of 16 batches/shift. If all pickers and batches are to filled to capacity, there are 480 batches and 1440 orders (the maximum batch size is thus 3), and the number of joint bin-packing and order combinations is

$$C_{total} = \begin{pmatrix} 1440 \\ 3 \end{pmatrix} \sum_{i=0}^{29} \begin{pmatrix} 480 - i16 \\ 16 \end{pmatrix}, \tag{6.9}$$

which is a very big number, approximately 10^{152}. To ground this number somewhat, the total number of atoms in the universe is estimated to be 10^{82}. Optimally solving the joint problem is impractical even for a small number of pickers.

6.3 Solving the Joint Batching and Generalized Assignment Problem

6.3.1 Routing Heuristic

To calculate total batch time, total travel distance of a picking tour is needed. The travel distance is assumed to be independent of the picker. The sample warehouse has multiple drop-off locations (more than one of which can be visited during a tour) and a middle cross-aisle (see Figure 3.1). The drop-off locations can only be visited after all items of an order have been collected. Each aisle can only be traveled in a single direction, while the cross-aisles are bidirectional. Tours start and end at the depot.

The number of potential batch evaluations for the problem introduced in Section 6.1 is potentially very large. For each batch evaluation, the routing problem needs to be solved. To calculate total travel distance, a computationally light heuristic is introduced.

In Algorithm 6, a version of the aisle-by-aisle routing heuristic (Vaughan, 1999) for warehouses with unidirectional travel in the aisles is presented. In this version of the algorithm, aisles do not need to be traveled completely if there are no picks past the middle aisle. It also incorporates drop-offs before the eventual return to the depot. The basic idea is to take advantage of the unidirectional travel in the warehouse and the middle cross-aisle. Once a path to pick all order lines has been formed (steps 1-4 and 6-8), the tour is optimally completed by adding the drop-off locations with Dijkstra's algorithm.

```
Input: A batch $r$, order drop-off locations $\mathcal{L}_r$, depot $D$, distance matrix
       $B$, set of aisles $\mathcal{A}$
Output: A tour $y$, travel distance $d_r$
1  Sort all orders in $r$ according to the aisle index.
2  Make a vector $\boldsymbol{v}_a$ of corresponding order lines for each relevant aisle
   $a \in \mathcal{A}$ to batch $r$.
3  Sort each aisle vector $\boldsymbol{v}_a$ according to the travel direction of the aisle.
4  Insert all vectors from the smallest aisle index to the largest index to a
   path $y_{small}$.
5  Use Dijkstra's algorithm to make $y_{small}$ into a tour by adding $\mathcal{L}_r$ and $D$
   to it.
6  Store the total path of $y_{small}$ cost in $d_{small}$ using $B$.
7  As in 4 but iterate from big to small aisle index and store the path in
   $y_{big}$.
8  Store the total path of $y_{big}$ cost in $d_{big}$ using $B$.
9  Use Dijkstra's algorithm to make $y_{big}$ into a tour by adding $\mathcal{R}_r$ and $D$
   to it.
10 if $d_{big} < d_{small}$ then
11     $y \leftarrow y_{big}$
12     $d_r \leftarrow d_{big}$
13 else
14     $y \leftarrow y_{small}$
15     $d_r \leftarrow d_{small}$
16 end
```

Algorithm 6: Middle aisle multi-dropoff routing heuristic

6.3.2 Initial Assignment of Batches

In many warehouses, the current practice is to assign the next batch in the queue to a picker who is first available to execute it. No planning or scheduling is involved. This method of assigning jobs to pickers is henceforth called the *first-free* assignment method. It assumed that first-free is used in the example warehouse.

Before assigning batches, an initial feasible solution must be found. This is done with a combination of a batching and an assignment algorithm. As the batching algorithm, either C&W(i) (Clarke and Wright, 1964) or VNS (Albareda-Sambola et al., 2009) is used. For assigning batches to pickers, first-free is applied.

6.3.3 Assignment of Customer Orders Based on Average Picker Productivity

Based on the data, the average picker productivity can be calculated using (6.7). This statistic is tracked in many warehouses already. A warehouse manager can choose to prioritize assigning work to the most productive pickers. To reflect this, and to try to justify the benefits of skill based assignment, the following heuristic, *fastest-first*, is used to assign work to the most productive pickers first. It will usually result in the slowest pickers being left out of the assignment. This algorithm assumes that there are as many pickers available as needed.

Step 1. Batch all orders based on travel distance using VNS.

Step 2. Sort all available picker based on productivity in descending order.

Step 3. Sort all batches based on the number of order lines in descending order.

Step 4. Assign the first unassigned batch to the picker who has the highest productivity value and so that the picker's maximum worktime is not exceeded. Repeat until all batches are assigned, otherwise exit.

6.3.4 Adaptive Large Neighborhood Search Algorithm

After forming a feasible initial solution, Adaptive Large Neighborhood Search (ALNS) (Ropke and Pisinger, 2006a) is used to search for good local optima. Pisinger and Ropke (2010) offer guidelines for designing ALNS algorithms, which have been mostly followed in this section. The ALNS algorithm uses a set of neighborhood heuristics $\mathcal{H}$. For the ALNS presented in this section, $\mathcal{H}$ is composed of five different heuristics. See Section 2.2.6 for more details of ALNS heuristics.

Following Ropke and Pisinger (2006a), a simulated annealing-type hill climbing scheme is implemented to complement the search. This simulated annealing scheme uses a geometric cooling schedule (Cohn and Fielding, 1999). At each iteration, the temperature is updated by $T \leftarrow \phi T$, where T is the temperature variable and ϕ the cooling coefficient.

A vector collecting all the adjustable parameters of the ALNS algorithm is defined as $\boldsymbol{v} = (\sigma_1, \sigma_2, \sigma_3, \lambda, \delta, \phi)$. The actual parameter set is given in Section 6.4.1.

Next, the neighborhood search heuristics are detailed. All of them maintain the feasibility of the solution by enforcing $t_{w,\mathcal{R}_w} \leq M_w$, where $t_{w,\mathcal{R}_w}$ is the total time to pick the set of batches $\mathcal{R}_w \subseteq \mathcal{R}$ assigned to picker w and M_w is the maximum working time for picker w. Only those solutions that are feasible according to the model in Section 6.1 are accepted. These heuristics are used to minimize total batch execution time, and an allocation can result in some of the pickers being left without work.

1 Random destroy, random repair

Step 1. Randomly choose $Q \sim U(2,7)$ batches to be destroyed from any of the pickers, resulting in the set of $\mathcal{O}_d$ customer orders, where $|\mathcal{O}_d| \geq Q$.

Step 2. Form a set $\mathcal{W}_d$ of pickers currently assigned to $\mathcal{O}_d$.

Step 3. Randomly choose a number of orders up to the maximum batch size of N orders from $\mathcal{O}_d$. Form a new batch from the chosen orders and assign it to a random worker $w \in \mathcal{W}_d$ if constraints (6.3) can be upheld. Remove the chosen orders from $\mathcal{O}_d$..

Step 4. Repeat Step 3 until there are no orders left to assign.

Step 5. Calculate the tour costs using Algorithm 6 for the new batches and form completion time forecasts for each new picker batch pair.

2 Savings-based destroy, random repair The aim of this heuristic is to destroy those batches that have the least savings value (the time difference of picking each order separately minus picking them together in a batch). Let s be a current solution to the BatchGAP and $S_{tot} \in \mathbb{R}$ be the sum of time-savings of all batches in s. Furthermore, let $S_{max} \in \mathbb{R}$ be the maximum batch savings value in s, and S_r be the savings of batch $r \in \mathcal{R}_s$. Then the probability of destroying a batch r is

$$p_r = (S_{max} - S_r)/S_{tot}. \tag{6.10}$$

Step 1. Calculate the batch savings for all batches in s.

Step 2. Use the roulette wheel method from (2.16) with batch probabilities calculated from Equation (6.10) to choose $Q \sim U(2,7)$ batches to destroy from any of the pickers, resulting in $M \geq Q$ total customer orders.

Step 3. Form a set $\mathcal{W}_d$ of pickers currently assigned to the batches to be destroyed.

Step 4. Randomly choose a maximum of N orders from the destroyed batches, form a new batch and assign it to a random worker $w \in \mathcal{W}_d$ if constraint (6.3) can be upheld.

Step 5. Repeat Step 4 until there are no orders left to assign.

Step 6. Get the tour costs for the new batches and form completion time estimates for each new picker batch pair.

3 Move batches This heuristic tries to move a set of batches $\mathcal{T} \subseteq \mathcal{R}$ between pickers while leaving the order composition of the batches intact. A batch $r \in \mathcal{T}$ from picker $w_i \in \mathcal{W}$ is moved to another picker $w_j \in \mathcal{W} - \{w_i\}$. No routing of batches is necessary, the work time estimates of the involved pickers just need to be calculated.

4 Empty picker and greedy repair With this heuristic, all batches are moved from a single picker to other workers. First, the picker is selected randomly from all pickers with batches. Then, each of the selected picker's batches is iteratively assigned to the picker who based on the forecast will execute it the fastest, and so that the capacity constraints for each picker are upheld.

5 Move all batches from a picker to another This final heuristic moves all batches from one random picker to the picker that executes them the fastest.

Acceptance of new solutions

After applying a neighborhood search heuristic to a current solution s, a new solution s' is found. Let s^* be the best solution found so far. Furthermore, let $f(s)$ be the cost of solution s. If $f(s') < f(s^*)$, the solution is accepted setting $s^* = s'$ and $s = s'$ The current score of the active heuristic is updated by $\Psi_h = \Psi_h + \sigma_1$. Otherwise, if $f(s') < f(s)$, the solution is accepted as the current best one setting $s = s'$ and $\Psi_h = \Psi_h + \sigma_2$. A worse solution than the current one can be accepted by the simulated annealing rule. Let $Q \sim U(0,1)$ and T be the current temperature of the algorithm. If $Q \leq \exp(\frac{f(s')-f(s)}{T})$, a hill climb is performed and setting $s = s'$ and $\Psi_h = \Psi_h + \sigma_3$.

6.4 Results

In this section, it is shown how original batch times relate to the modeled ones and compute new batching, routing and assignment solutions which are compared to the original ones. All algorithms were coded in C++ and run single-threaded on an Intel Xeon 3.2 GHz with 16GB of memory.

6.4.1 Parameter Calibration

Following Ropke and Pisinger (2006a), the calibration parameter vector is initialized to $\boldsymbol{v} = (\sigma_1, \sigma_2, \sigma_3, \lambda, \delta, \phi) = (33, 13, 9, 0.95, 200, 0.999995)$ with the exception of δ, which was not used by these authors; its value was set according to Gharehgozli et al. (2013). In order to calibrate the parameters, two sample instances were run of 3 pickers with 78 orders, and 10 pickers with 360 orders. One parameter was varied at a time on a uniform interval for each parameter around the initial value. The best performing parameters were chosen and set as $\boldsymbol{v} = (65, 13, 9, 0.95, 91, 0.999995)$. The scaling factor of the computational complexity of heuristic h, ζ_h from (2.18), is not used, as no significant improvement in solution quality or time was found when it was used. The starting temperature T_0 is set such that the probability of an uphill climb is 0.5 if the new solution is 3% worse than the sum of batch execution times in the starting solution s_0 by $T_0 = -0.03 f(s_0)/\ln 0.5$.

6.4.2 Data Preparation

During a daily 8h shift in the sample warehouse, the number of available pickers can vary between 20 and 40. Since the data do not contain due times for the batches, and only a part of all the pickers appearing in the data can be used due to data cleaning, comparing the allocations and batches of the real shifts and the ones reconstructed by the ALNS is not necessarily meaningful. To still allow a comparison, virtual days are constructed by partitioning the data of 24,669 batches into 12 (about) equally sized subsets. Each partition forms one virtual day, resulting in approximately 2056 orders (685 batches) per day. A picker qualifies for inclusion in the workforce of the virtual day if both the sum of real execution times of the batches he or she performed and the sum of the forecast batch execution times exceed the minimum threshold of M_{min}. This is done to guarantee that each picker included carries out sufficient work. The total execution time of all the batches executed by a picker should not exceed the maximum threshold time M_{max}. If adding a batch were to exceed this threshold, it would be discarded for this picker.

The used data only contain Hamiltonian Paths starting from the first pick line to the last line, leaving out the roll container drop-offs and eventual return of the picker to the depot. Each order has its own unique drop-off location. Figure 3.1 shows the different locations of these drop-offs. Dijkstra's algorithm is applied to find the combined minimum distance of visiting the order drop-off locations and further traveling from the last drop-off location to the depot and from the depot to the first pick location. Using this distance, the extra time to complete the pick tour is now calculated, assuming an average truck speed of 1 m/s, and taken into consideration when calculating sums of batch execution times in constructing a virtual day.

Now the real batch execution times and their forecasts for these newly created virtual days are known. These can be used to compare the ALNS batching, routing, and assignment results with the original solutions.

6.4.3 Experimental Setup

Virtual days are constructed as detailed in Section 6.4.2. For each virtual day, the constraints in (6.3) are set to 8h. Two different instances are run by setting M_{min} to 7.15h or 7.4h and M_{max} to 7.75h in both real time and forecast time. This results in 29 and 20 pickers working on the virtual days on average, respectively. Out of the possible 99 pickers, 93 qualify for the simulations with $M_{min} =$ 7.15h and 86 for $M_{min} =$ 7.4h. The ALNS is run for 5,000,000 iterations in both cases.

Work is currently issued according to the first-free allocation. Two batching algorithms, C&W(i) and the Variable Neighborhood Search (VNS) by Albareda-Sambola et al. (2009), are used to generate an initial solution for the ALNS. C&W(i) provides a quick solution to the batching problem, while the VNS method is shown to provide good results with the tradeoff of exploring large parts of the solution space (Albareda-

Sambola et al., 2009, Matusiak et al., 2014). Using these initial solutions, the ALNS is used in two ways: (1) including all neighborhood search heuristics, and (2) allowing only those heuristics which do not break down batches (i.e., excluding heuristics 1 and 2).

In the ALNS heuristic presented in this chapter, a computationally light routing heuristic (see Section 6.3.1) is used, as it has to be invoked numerous times. In order to understand how far this routing is off a near heuristic, it is compared with LKH (Helsgaun, 2000, Theys et al., 2010) for 10,000 random batches constructed from customer orders from the data-set. LKH has a gap of about 0.1% with optimal routing (Theys et al., 2010), at the expense of much longer computation times. LKH does not handle precedence constraints, so drop-offs are left out of the tours for this comparison. That is, in this experiment the tours of both the light routing heuristic LKH lack precedence constraints. On average, LKH leads to solutions of about 11.7% shorter tours.

6.4.4 Time Savings and Comparison to Original Solution

In this section, the algorithmic solutions are compared to the original and initial batching allocation, as well as to each other. The following abbreviations are used for the solutions in Figures 6.1 and 6.2. Unless otherwise noted, batch execution times are forecast:

BF batching done with VNS and allocation using *fastest-first*;

CWI batching done with C&W(i) and allocation using first-free;

CWR improvement of CWI using the ALNS algorithm with no further rebatching (only heuristics 3, 4, and 5 are used);

CWF as CWR, but with all neighborhood search heuristics used in the ALNS;

VNSI batching done with VNS and allocation using first-free;

VNSR improvement of VNSI using the ALNS algorithm with no further rebatching (only heuristics 3, 4, and 5 are used);

VNSF as VNSR, but with all neighborhood search heuristics used in the ALNS.

The comparison baselines in figures 6.1 and 6.2 are the forecast batch execution times with real batches during a virtual day. Both figures are

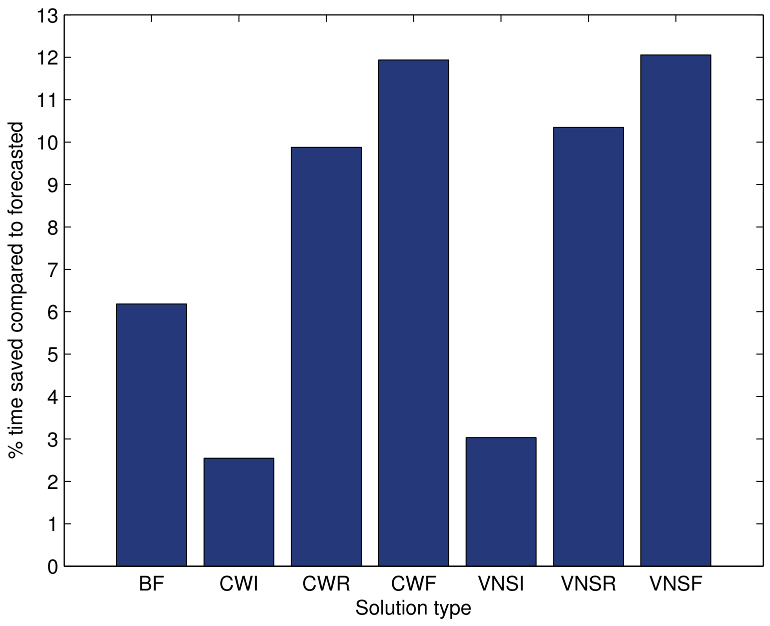


Figure 6.1. Savings generated by different algorithms, compared to the original batching with forecast execution times during a virtual day. 7.15h minimum time to qualify, 29 pickers. The real batch execution times for the original batching are 3% smaller than the forecast in this case.

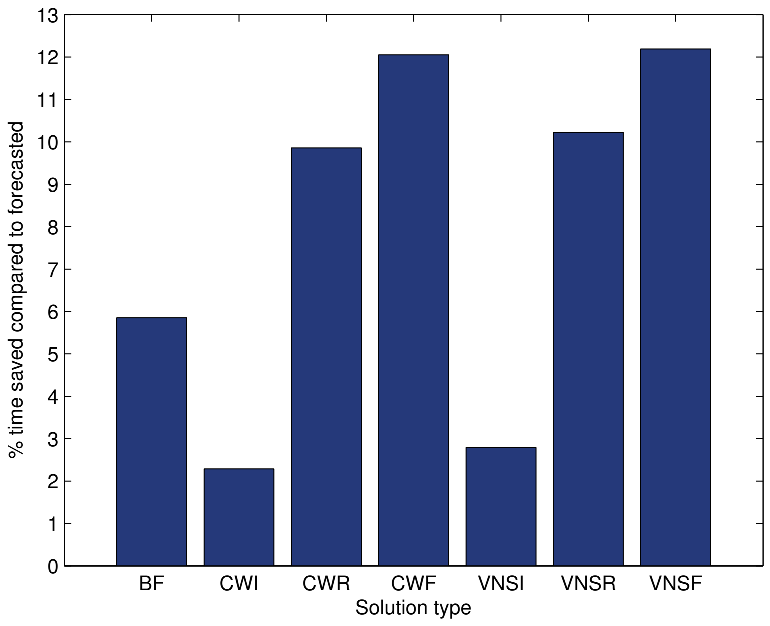


Figure 6.2. Savings generated by different algorithms, compared to the original batching with forecast execution times during a virtual day. 7.4h minimum time to qualify, 20 pickers. The real batch execution times for the original batching are 1% smaller than the forecast in this case.

quite similar, so the amount of spare time in the virtual day set-ups or the number of pickers does not affect the performance of the algorithms much. The forecast in Figure 6.1 gives a pessimistic estimate, since the real times are about 3% smaller. In Figure 6.2 the real times are 1% smaller. If jobs are assigned based on the average productivity (category BF), savings of around 6% can be achieved in both cases. The initial solution CWI gives improvements of over 2% compared to forecast batch execution times with real batches. The best solution is provided by the combination of VNS batching and the ALNS, i.e., solution VNSF. However, CWF is very close in terms of solution quality and much faster to execute. In the 29-picker case, CWF took on average 80 minutes to solve the problem of batching and assigning 2048 orders, whereas VNSF took 120 minutes. For the 20-picker case, these times are 35 minutes and 70 minutes, respectively. In the sample warehouse, orders arrive early in the morning a couple of hours before the morning shift starts, so there is ample time to run the algorithm. Both figures show that it is beneficial to rebatch using the neighborhood search heuristics 1 and 2 in addition to 3, 4, and 5, which just move batches between pickers.

Most importantly, Figures 6.1 and 6.2 show that significant savings in total batch execution time can be obtained by assigning the right batches to the right pickers. Almost 10% more time can be saved when comparing VNSF to CWI and to VNSI. Compared to the forecast and real time for the original allocation, between 10% and 12% can be saved. Finally, when comparing VNSF with BF, i.e., assigning based on picker skill vs. on productivity, savings of 6% in total batch execution time can be achieved.

6.4.5 Impact on Picker Productivity

Figures 6.3 and 6.4 show the effect of the VNSF solution compared to a VNSI solution, which implies state-of-the art batching with a first-free picker assignment. Pickers are divided into three categories based on their productivity (lines picked per time unit worked) in the VNSI solution: the slowest 20%, the medium 60%, and the fastest 20%, with a total of 20, 53, and 20 pickers, and 15 pickers (19, 48, and 19 in Figure 6.4), respectively. However, as noted below, the slowest pickers receive much less work. Here productivity is calculated by the total number of assigned lines divided by the total time to process all assigned batches, i.e., actual work time.

All categories improve productivity, while most improvements can be ob-

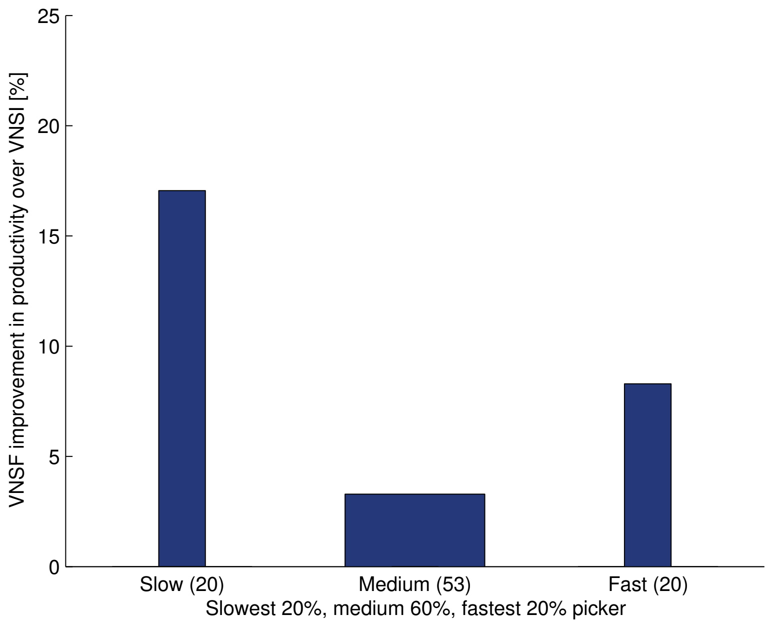


Figure 6.3. Change in average productivity by picker category. 7.15h minimum time to qualify, 29 pickers

served in the slow and medium picker categories. For pickers in the slow, medium, and fast categories for the case of 29 pickers (Figure 6.3), the initial average productivity values are 1.14, 1.43, and 1.81 lines/minute, respectively. The VNSF solution improved productivity by 19%, 7%, and 12% on average compared to the VNSI for the same categories. For the 20 picker case, VNSI productivities are 1.12, 1.44, and 1.83 lines/minute, which improve by 17%, 5%, and 15% (Figure 6.4). The improvement in all categories is a result a reduction in total working time compared to the initial allocation as picker skills are taken into consideration. Additionally, some pickers, particularly those in the slow category, are assigned less work, and in many cases do not get to work full shifts.

Since VNSI is used as a reference, this improvement in productivity must result from better picker assignment. However, the slowest pickers receive fewer orders to be picked, resulting in 40% to 60% fewer pick lines, as can be seen from figures 6.5 and 6.6. The medium and fast pickers tend to get more work in both the 29- and the 20-picker cases.

To better understand which of the pickers are included in the ALNS assignment over the entire horizon of eight virtual days, the picker categories used above are divided into subcategories, indicated by two characters, XY. X stands for the main categories S, M, F (Slow, Medium, Fast),

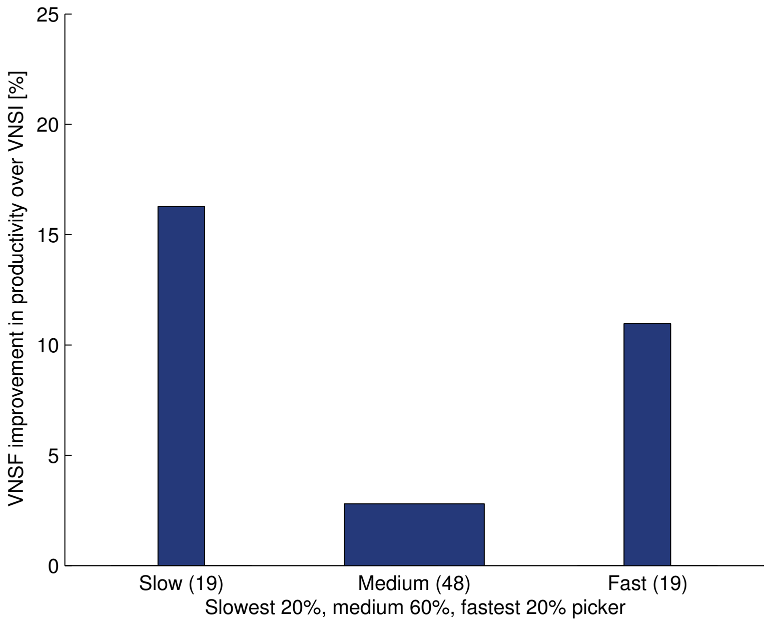


Figure 6.4. Change in average productivity by picker category. 7.4h minimum time to qualify, 20 pickers

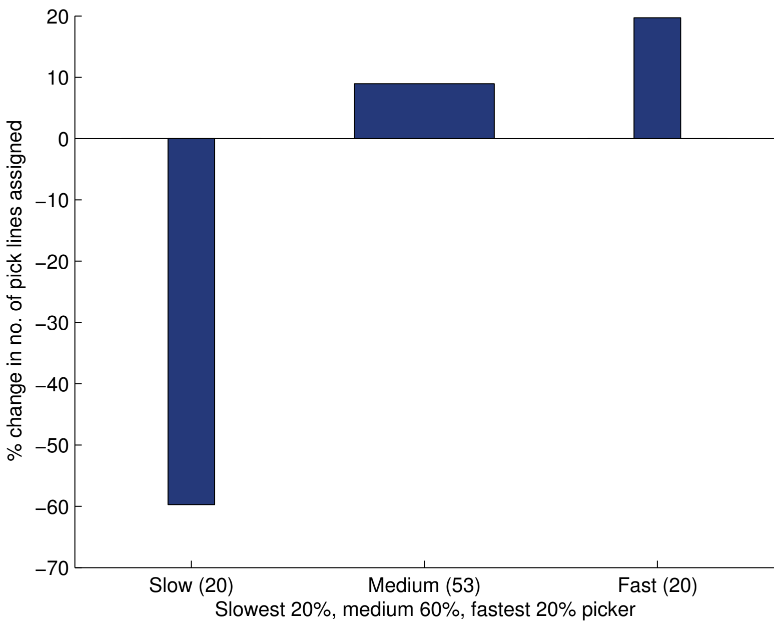


Figure 6.5. Change in number of lines assigned by picker category. 7.15h minimum time to qualify, 29 pickers.

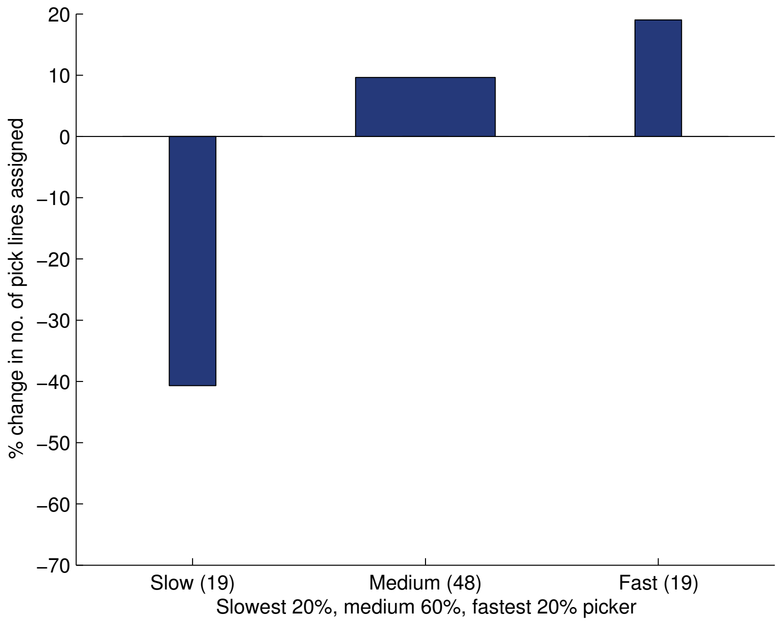


Figure 6.6. Change in number of lines assigned by picker category. 7.4h minimum time to qualify, 20 pickers.

based on the productivity in the VNSI solution. Y has three categories, 0, 1, 2, indicating whether the picker: is left out of the assignment completely, or receives less than a fifth of his or her total number of order lines assigned in the VNSI solution, or receives more than a fifth of his or her total number of order lines assigned in the VNSI solution, respectively.

Figures 6.7 and 6.8 show the division of pickers over these categories compared to VNSI. The pickers in the slow category are the most affected: in the 29-picker per day case, two out of 15 pickers are completely left without work for all eight days. In the 20-picker case, one picker is left out. In the medium category, five and four pickers left out of the assignment, whereas all pickers in the fast category are assigned to work. Apparently, it pays off not to use some of the workers at all and to use some of the workers as little as possible.

6.4.6 Effect of Skill on Batch Assignment

Forecasting the time for a picker to pick a batch is done by using the model described in Section 5.3. By including the knowledge of the batch parameters in the optimization process of the joint batching and generalized assignment problem, a better result can be achieved than optimizing by

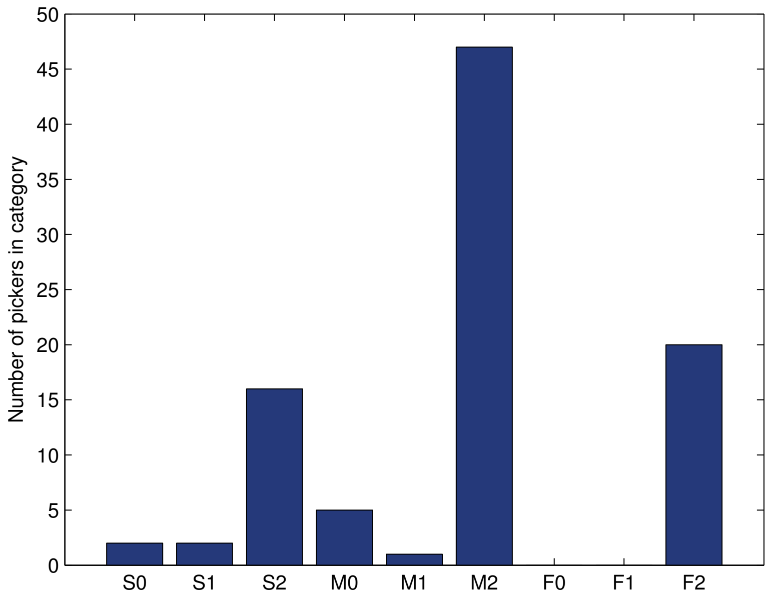


Figure 6.7. Picker categories after final assignment. 7.15h minimum time to qualify, 29 pickers on average.

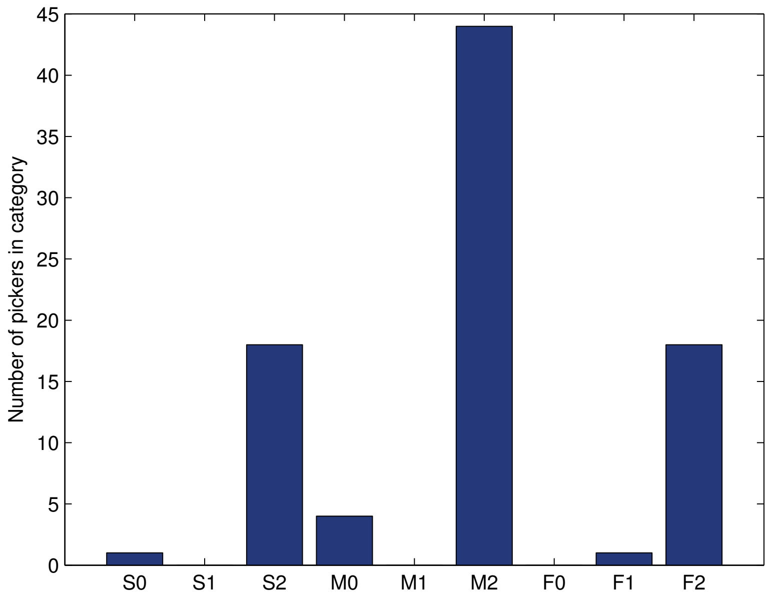


Figure 6.8. Picker categories after final assignment. 7.4h minimum time to qualify, 20 pickers on average.

travel distance and statistical productivity (picking speed). The productivity of a picker is a statistic already tracked in many picking processes. However, to the author's knowledge, it is not extensively used in assigning work: but neither are other statistics. It is common to treat order pickers as a homogeneous unit.

Setup This following experiment tries to provide an insight to how much total order picking time can be saved by taking into account the additional information of batch parameters and picker models when compared to good travel distance-based batching and assigning work based on the pickers' productivity. If slow pickers are left out, it will improve the overall performance of the process, as seen previously in sections 6.4.4 and 6.4.5.

To start, 12 virtual days with a 7.4h initial threshold are constructed, similarly as explained in Section 6.4.2. Based on this, a combination of two algorithms is used to optimize the order batching and to assign work. For the batching, the state-of-the-art VNS algorithm of Albareda-Sambola et al. (2009) is used. *Fastest-first*, which optimizes total batch time based on the most commonly measured statistic of picker productivity, is applied to assign batches to the most productive pickers. This setup will leave out the least productive pickers. Finally, if any pickers exist that work for less than 7.4h after the *fastest-first* assignment, they are left out of this experiment. This results in an allocation where only the statistically most productive pickers are selected, with each of the pickers being assigned a minimum of 7.4h of work. It is relatively tight in terms of free time so that not many pickers will be left out when improving it with the ALNS.

After constructing the initial allocation with the VNS batching and *fastest-first* assignment, the ALNS algorithm is run to improve upon this initial solution. Both the ALNS and *fastest-first* try to minimize total batch execution time over all batches and thus are comparable. The resulting allocation takes into account the additional information of batch parameters and picker coefficients that affect the output time.

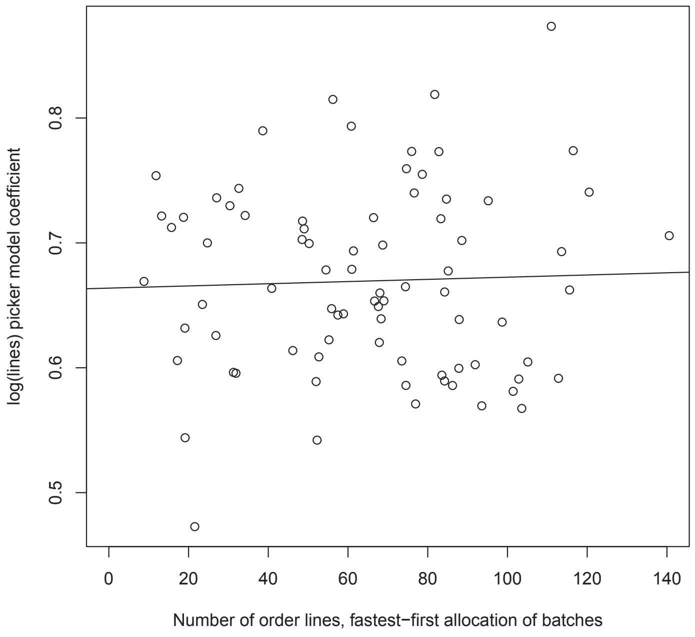
Results The average number of qualifying pickers per 2048-order virtual day is 19. Total time savings after applying the ALNS are over 5.5%. As a result of applying the ALNS, two pickers of the 80 that participate in these virtual days are always left out work: the first on two separate days, and the second on one day.

On average, productivity of the remaining pickers increases by 4.6%. The average time to complete an order line is decreases from 39.8s to

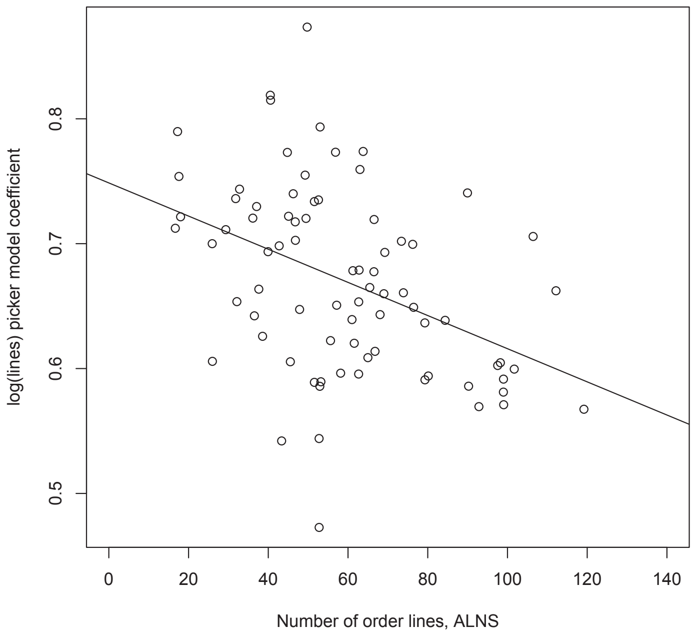
37.8s, or 5.5%. As picker productivity is defined as the number of order lines picked per time unit in (6.7), its increase fails to explain the savings completely. Thus it can be concluded that 0.9% of the time savings result from leaving out the two pickers. The increase in picker productivity must therefore come from a better assignment of orders (and batches) between the participating pickers.

To illustrate how orders and batches are assigned to the most suitable pickers, the batch parameters (i.e., *Lines*, *Travel*, etc.) of each batch assigned to the pickers are logged for both the initial allocation constructed with *fastest-first* and the final allocation after applying the ALNS. These data are then used to calculate the average batch assigned to each picker.

A low forecasting model coefficient corresponds to that particular picker being skilled in that category (see Section 5.3). The hypothesis was made that savings that are found by applying the ALNS-algorithm are at least in part due to difference in picker skill: pickers with low coefficients in a particular category will generally be assigned batches with high values in the same category. For example, if a picker is more skilled in picking heavy items, i.e., he/she has a low model coefficient in $ln(Mass)$, and he/she should be generally assigned heavier batches. For each of the model inputs (batch parameters), this hypothesis can be verified from figures 6.9-6.13. In these figures, each of the model coefficients of the participating pickers is plotted against the average batch parameter of the batches that are assigned to those pickers. Figures 6.9 show how $ln(Lines)$ relates to the number of lines of in a batch and figures 6.10 show the relation of $ln(Travel)$ to the average travel distance of a corresponding batch, and so on. In the top figures, marked **(a)**, batch assignment is done with *fastest-first*, and in the bottom ones, marked **(b)**, ALNS is applied to the BatchGAP. The line in each of the figures shows an OLS fit with the picker model coefficients as the dependent variables and the batch parameter as the independent variables. These figures indicate that the ALNS tends to assign batches with high values in a particular parameter to those pickers with a corresponding low model coefficient, while the combination of VNS order batching and *fastest-first* does not take into account skill. Most of the savings gained from applying the ALNS result from assigning the right orders and batches to the right pickers, as they cannot come solely from leaving out the slowest pickers.

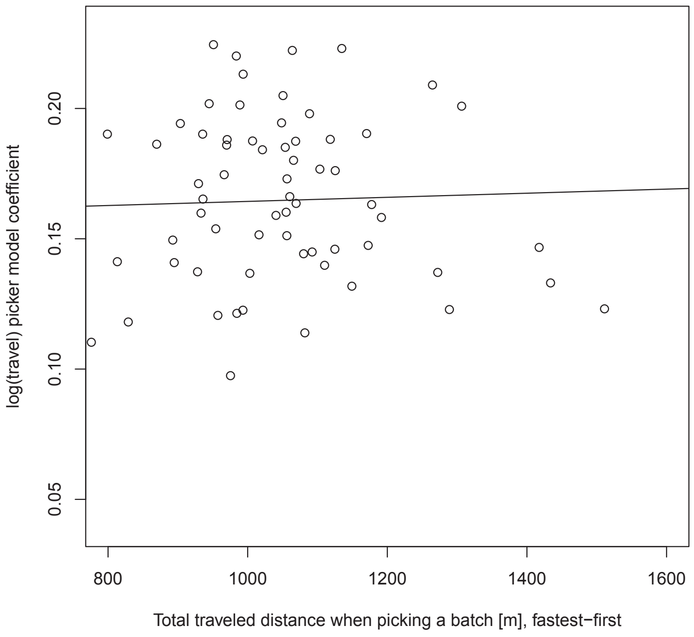


(a) VNS batching and fastest-first assignment

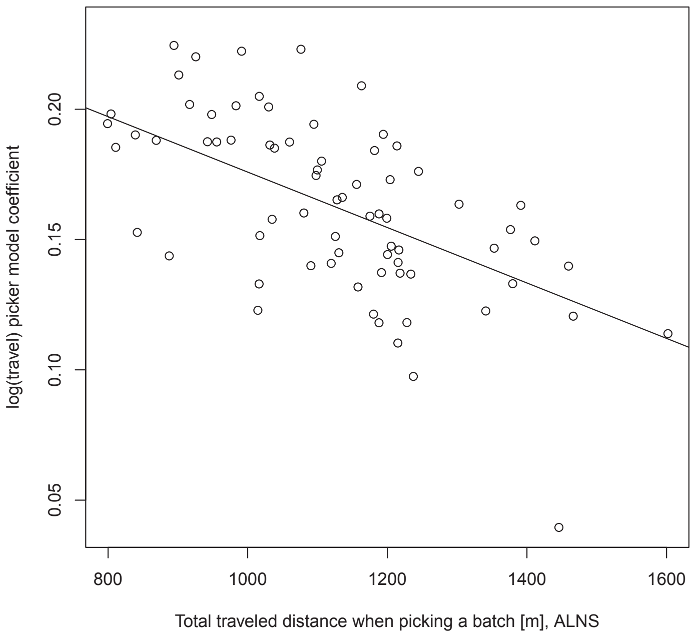


(b) ALNS

Figure 6.9. The average number of order lines in a batch assigned to a picker vs. the corresponding $ln(Lines)$ forecasting model coefficient of that picker, for two different ways of solving the BatchGAP.

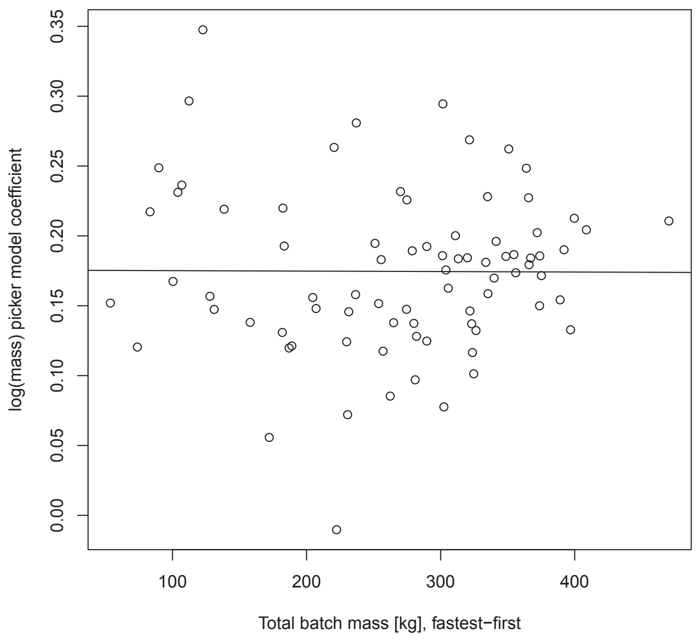


(a) VNS batching and fastest-first assignment

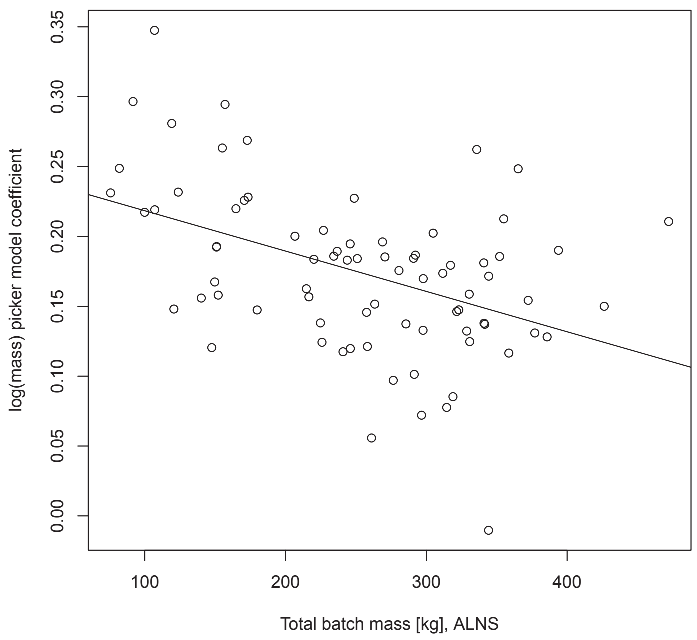


(b) ALNS

Figure 6.10. The travel distance to pick a batch assigned to a picker vs. the corresponding $ln(Travel)$ forecasting model coefficient of that picker, for two different ways of solving the BatchGAP.

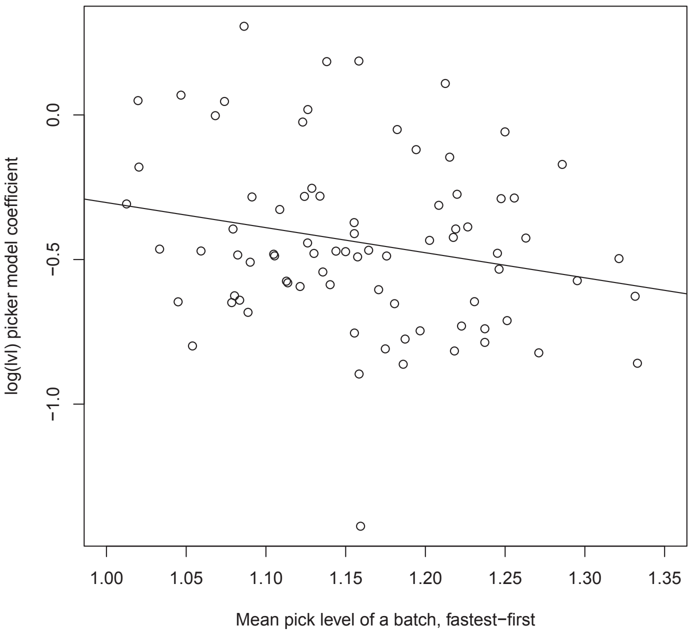


(a) VNS batching and fastest-first assignment

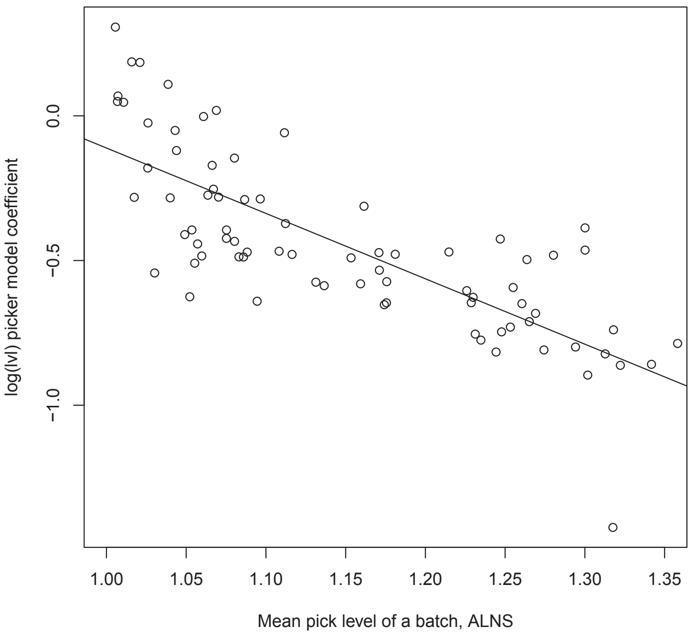


(b) ALNS

Figure 6.11. The average mass of a batch assigned to a picker vs. the corresponding $ln(Mass)$ forecasting model coefficient of that picker, for two different ways of solving the BatchGAP.

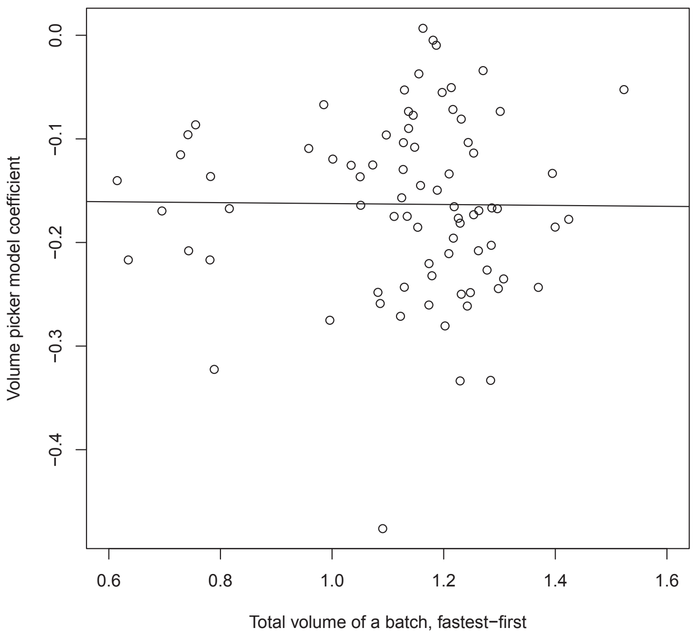


(a) VNS batching and fastest-first assignment

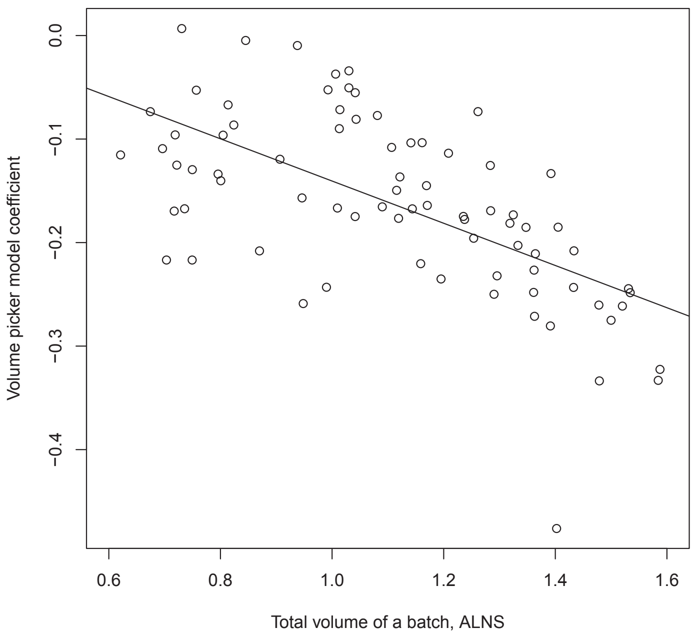


(b) ALNS

Figure 6.12. The mean pick level of a batch assigned to a picker vs. the corresponding $ln(Level)$ forecasting model coefficient of that picker, for two different ways of solving the BatchGAP.



(a) VNS batching and fastest-first assignment



(b) ALNS

Figure 6.13. The average volume of a batch assigned to a picker vs. the corresponding Vol forecasting model coefficient of that picker, for two different ways of solving the BatchGAP.

6.5 Discussion

This chapter introduces a method to model and solve the combined batching, routing, and picker assignment problem, based on properties of the batches to be executed. The method consists of two parts: first the batch execution times are estimated with multilevel analysis based on properties of historical batches, such as total number of lines, distance between items, total mass and volume, and pick level. Second, the integrated batching, routing, and picker assignment problem is solved using an ALNS heuristic.

The method is tested for a large retail warehouse for which three months of complete pick data could be obtained. In Chapter 5, multilevel modeling of the pickers shows that 13.1% of the variance in batch execution time can be explained by differences among pickers. Subsequent application of the ALNS heuristic shows that improvements of 9% in total batch execution time can be achieved compared to state-of-the art batching with VNS, but ignoring picker skills. Compared to the current picker assignment, 12% is saved depending on the instance (see figures 6.1 and 6.2).

The presented method is generally applicable in picker-to-parts warehouses, independent of the layout. However, the choice of the routing heuristic should be made according to the case. The main requirement for applying the method is the availability of historical pick data to build the forecasting models of the pickers. These data are gathered in most modern warehouses, by pick-by-voice, pick-by-light, and pick-by-terminal systems. Using these data, warehouse management systems can easily be extended by including forecast pick times in the batching and assignment of batches.

Although the method was tested in only one warehouse, it is expected that savings can be achieved in other picker-to-parts picking facilities. The magnitude of the savings will depend on several factors, such as the number of order lines in a batch (more lines yield a higher saving opportunity), the size of the pick force (more workers means more opportunity for savings), the product characteristics (larger variety in sizes and volume requires more stacking skills of the workers and potentially larger saving opportunities), and variation in the height levels at which the picking takes place.

This approach has some limitations. The data obtained are extensive, but they do not contain the departure times of trucks shipping the orders.

Therefore, there is no information about time windows for the orders, which are to some extent included in the real batch data. This means the calculated savings slightly overestimate the real savings as the search space for selecting the next orders to the batch shrinks. However, this only affects savings compared to the original batching and allocation.

The results presented here suggest that the impact of the right worker is over 9% in execution time, or 6% if picker productivity is taken into account when assigning work. Furthermore, it is also shown that the skill of a picker in a particular category influences the type of batches he or she is assigned: more skilled pickers tend to get more challenging batches to execute. These results imply that managers should consider employee skills and capabilities during the decision making process. Currently worker to work assignment is primarily based on qualifications. However, when multiple workers qualify for the job, individual differences might still be exploited. Currently in most warehouses, when workers are qualified for the same work, the next job is assigned to the first available worker. It may pay off to select the right job from the stack to be executed by the best skilled worker. In addition, when hiring a flex workforce and past performance data are available, the warehouse manager can use them to hire the best performing pickers given a set of orders.

7. Conclusions

This thesis has two main contributions to the current state-of-the-art in warehousing and operations research. First, precedence-constrained routing is considered in conjunction with the order batching problem. This type of routing introduces new requirements for the order batching algorithm, as it is computationally heavy. It is possible to estimate savings of batches composed of precedence-constrained customer orders quite accurately. In other words, the estimate is able to predict how beneficial it is to combine sets of cities into a larger Traveling Salesman Problem. This is the key contribution of Chapter 3, as it leads to a dramatic reduction in calls to routing algorithms, which are integral subroutines of most batching algorithms. The algorithm proposed in Chapter 3, Precedence-Constrained Estimated Savings-based batching, compares favorably in solution quality and execution time to state-of-the-art batching algorithms and to optimal solutions for small instances. As seen in Chapter 4, it is also possible to swap the routing algorithm used for another one.

The second main contribution is the study of modeling pickers and assigning the right orders and batches to the right pickers, presented in Chapters 5 and 6. To the author's knowledge, it is the first work that models directly how job parameters affect workers' job execution times (the worker skills) and exploits this knowledge directly when assigning work. The main goal of this study is to show that it is beneficial to consider picker skills in order picking models. A simulation experiment shows that a total of 9% in total order picking can be saved if the order pickers' skills are taken into account in conjunction with batching orders.

This study presents multiple opportunities for further research. First, the results should be further validated in warehouses of different types, with varying layouts, order, batch and product properties, and those that use incentives for pickers as motivation. To form realistic picker profiles, data logs need to be obtained, which can be difficult to obtain from companies. The data can be noisy and contain outliers. Additionally, as is the case with this study, the data can have multiple picking modes, which may need to be handled separately.

Second, a straightforward extension to this method can be done with the inclusion of time windows. This would require the inclusion of additional constraints in the model and modifications to the neighborhood search heuristics. Another way of extending the method is to consider a rolling horizon setting, where orders and batches are assigned online to pickers.

Third, although the models of the pickers explain 86% of the variance, the forecasts might be made more accurate by including other behavioral variables. Based on previous studies (Bendoly et al., 2006, De Koster et al., 2011, De Vries et al., 2013), it is likely that also behavioral variables may have an additional effect. These can include motivational variables such as prevention and promotion focus, or personality traits.

Fourth, insights obtained from this study can be applied to other areas in addition to picker-to-parts order picking. Parts-to-picker systems, where products are picked by machines and delivered to the right picker for further processing, are a natural extension. In general, any job type that can be parameterized and for which the execution times can be accurately forecast based on these parameters for the different workers, qualifies for careful assignment. The classical vehicle routing problem is an example. Parcel carriers have to make many deliveries in different types of areas, urban as well as rural. Based on the types of deliveries, neighborhood, number of parcels, distances, and weight, etc., it might pay off to select the best driver to fit the job circumstances.

Bibliography

Ahuja, R. K., Orlin, J. B., Sharma, D., et al. (1998). *New neighborhood search structures for the capacitated minimum spanning tree problem*. Sloan School of Management, Massachusetts Institute of Technology.

Albareda-Sambola, M., Alonso-Ayuso, A., Molina, E., and De Blas, C. (2009). Variable neighborhood search for order batching in a warehouse. *Asia-Pacific Journal of Operational Research*, 26(05):655–683.

Bartholdi, J. J. and Eisenstein, D. D. (1996). A production line that balances itself. *Operations Research*, 44(1):21–34.

Bartholdi, J. J., Eisenstein, D. D., and Foley, R. D. (2001). Performance of bucket brigades when work is stochastic. *Operations Research*, 49(5):710–719.

Baumann, H. (2013). *Order picking supported by mobile computing*. PhD thesis, University of Bremen.

Beamon, B. M. (1998). Supply chain design and analysis: Models and methods. *International journal of production economics*, 55(3):281–294.

Bendoly, E., Donohue, K., and Schultz, K. L. (2006). Behavior in operations management: Assessing recent findings and revisiting old assumptions. *Journal of Operations Management*, 24(6):737–752.

Berger, S. M. and Ludwig, T. D. (2007). Reducing warehouse employee errors using voice-assisted technology that provided immediate feedback. *Journal of Organizational Behavior Management*, 27(1):1–31.

Bliese, P. D. (2002). *Multilevel random coefficient modeling in organizational research: Examples using SAS and S-PLUS.* Jossey-Bass.

Bowersox, D. J. (2011). *Supply chain logistics management*. Tata McGraw-Hill Education.

Bryk, A. S. and Raudenbush, S. W. (1992). *Hierarchical linear models: Applications and data analysis methods.* Sage Publications, Inc.

Campbell, G. M. and Diaby, M. (2002). Development and evaluation of an assignment heuristic for allocating cross-trained workers. *European Journal of Operational Research*, 138(1):9–20.

Cattrysse, D. G. and Van Wassenhove, L. N. (1992). A survey of algorithms for the generalized assignment problem. *European Journal of Operational Research*, 60(3):260–272.

Chan, F. and Kumar, V. (2008). A TSSA algorithm based approach to enhance the performance of warehouse system. In *10th IEEE International Conference on Control, Automation, Robotics and Vision, 2008. ICARCV 2008.*, pages 1696–1701.

Clarke, G. and Wright, J. (1964). Scheduling of vehicles from a central depot to a number of delivery points. *Operations Research*, 12(4):568–581.

Cohn, H. and Fielding, M. (1999). Simulated annealing: Searching for an optimal temperature schedule. *SIAM Journal on Optimization*, 9(3):779.

Cordeau, J.-F., Laporte, G., Pasin, F., and Ropke, S. (2010). Scheduling technicians and tasks in a telecommunications company. *Journal of Scheduling*, 13(4):393–409.

Cornuéjols, G., Fonlupt, J., and Naddef, D. (1985). The traveling salesman problem on a graph and some related integer polyhedra. *Mathematical Programming*, 33(1):1–27. Available from: `http://dx.doi.org/10.1007/BF01582008`.

Coyle, J., Bardi, E., and Langley, C. (1996). *The Management of Business Logistics*. West Publishing Company.

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

De Koster, R., Stam, D., and Balk, B. M. (2011). Accidents happen: The influence of safety-specific transformational leadership, safety consciousness, and hazard reducing systems on warehouse accidents. *Journal of Operations Management*, 29(7):753–765.

De Koster, R. and Van der Poort, E. (1998). Routing order pickers in a warehouse: a comparison between optimal and heuristic solutions. *IIE Transactions*, 30(5):469–480.

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

De Vries, J., De Koster, R., and Stam, D. (2013). Safety does not happen by accident: How to manage a safe warehouse? *Working paper, Erasmus University, Rotterdam*.

Dekker, R., De Koster, R., Roodbergen, K., and Van Kalleveen, H. (2004). Improving order-picking response time at Ankor's warehouse. *Interfaces*, 34(4):303–313.

Doerr, K. H. and Arreola-Risa, A. (2000). A worker-based approach for modeling variability in task completion times. *IIE Transactions*, 32(7):625–636.

Drury, J. (1988). Towards more efficient order picking. IMM Monograph 1, The Institute of Materials Management, Cranfield, U.K.

Duan, N. (1983). Smearing estimate: a nonparametric retransformation method. *Journal of the American Statistical Association*, 78(383):605–610.

Escudero, L. (1988). An inexact algorithm for the sequential ordering problem. *European Journal of Operational Research*, 37(2):236–249.

Fisher, M. L., Jaikumar, R., and Van Wassenhove, L. N. (1986). A multiplier adjustment method for the generalized assignment problem. *Management Science*, 32(9):1095–1103.

Gademann, N. and Van de Velde, S. (2005). Order batching to minimize total travel time in a parallel-aisle warehouse. *IIE Transactions*, 37(1):63–75.

Gelman, A. and Hill, J. (2007). *Data analysis using regression and multilevel/hierarchical models*. Cambridge University Press.

Gharehgozli, A., Laporte, G., Yu, Y., and De Koster, R. (2013). Scheduling twin yard cranes in a container block. *Transportation Science, forthcoming*.

Goetschalckx, M. and Ashayeri, J. (1989). Classification and design of order picking. *Logistics Information Management*, 2(2):99–106.

Gu, J., Goetschalckx, M., and McGinnis, L. F. (2007). Research on warehouse operation: A comprehensive review. *European Journal of Operational Research*, 177(1):1–21.

Hart, P., Nilsson, N., and Raphael, B. (1968). A formal basis for the heuristic determination of minimum cost paths. *IEEE Transactions on Systems Science and Cybernetics*, 4(2):100–107.

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

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

Hong, S., Johnson, A., and Peters, B. (2012a). Batch picking in narrow-aisle order picking systems with consideration for picker blocking. *European Journal of Operational Research*.

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

Hsieh, L. and Huang, Y. (2011). New batch construction heuristics to optimise the performance of order picking systems. *International Journal of Production Economics*, 131(2):618–630.

Juran, D. C. and Schruben, L. W. (2004). Using worker personality and demographic information to improve system performance prediction. *Journal of Operations Management*, 22(4):355–367.

Kohonen, T. (1990). The self-organizing map. *Proceedings of the IEEE*, 78(9):1464–1480.

Kreft, I. and De Leeuw, J. D. (1998). *Introducing Multilevel Modeling*. Sage, London, UK.

Kubo, M. and Kasugai, H. (1991). The precedence constrained traveling salesman problem. *Journal of the Operations Research Society of Japan*, 34(2):152–172.

Kuijt-Evers, L., Bosch, T., Huysmans, M., De Looze, M., and Vink, P. (2007). Association between objective and subjective measurements of comfort and discomfort in hand tools. *Applied Ergonomics*, 38(5):643–654.

Larco Martinelli, J. (2010). *Incorporating Worker-Specific Factors in Operations Management Models*. PhD thesis, Erasmus Research Institute of Management, Erasmus University Rotterdam. Available from: `http://hdl.handle.net/1765/21527`.

Lundy, M. and Mees, A. (1986). Convergence of an annealing algorithm. *Mathematical Programming*, 34(1):111–124.

MacQueen, J. (1967). Some methods for classification and analysis of multivariate observations. In *Proceedings of the fifth Berkeley symposium on mathematical statistics and probability*, volume 1, page 14. California, USA.

Matusiak, M., de Koster, R., Kroon, L., and Saarinen, J. (2014). A fast simulated annealing method for batching precedence-constrained customer orders in a warehouse. *European Journal of Operational Research*, 236(3):968 – 977. Vehicle Routing and Distribution Logistics. Available from: `http://www.sciencedirect.com/science/article/pii/S0377221713004670`.

Miller, C., Tucker, A., and Zemlin, R. (1960). Integer programming Formulation of Traveling Salesman Problems. *Journal of the A.C.M.*, 7:326–329.

Nakagawa, S. and Schielzeth, H. (2013). A general and simple method for obtaining R^2 from generalized linear mixed-effects models. *Methods in Ecology and Evolution*, 4(2):133–142. Available from: `http://dx.doi.org/10.1111/j.2041-210x.2012.00261.x`.

Pentico, D. W. (2007). Assignment problems: A golden anniversary survey. *European Journal of Operational Research*, 176(2):774–793.

Pisinger, D. and Ropke, S. (2010). Large neighborhood search. In *Handbook of metaheuristics*, pages 399–419. Springer.

Powell, S. G. and Schultz, K. L. (2004). Throughput in serial lines with state-dependent behavior. *Management Science*, 50(8):1095–1105.

Psaraftis, H. (1980a). A dynamic programming approach for sequencing groups of identical jobs. *Operations Research*, 28(6):1347–1359.

Psaraftis, H. (1980b). Dynamic programming solution to the single vehicle many-to-many immediate request dial-a-ride problem. *Transportation Science*, 14(2):130–154.

Psaraftis, H. (1983). An exact algorithm for the single vehicle many-to-many dial-a-ride problem with time windows. *Transportation Science*, 17(3):351–357.

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

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

Roodbergen, K. and De Koster, R. (2001a). Routing methods for warehouses with multiple cross aisles. *International Journal of Production Research*, 39(9):1865–1883. Available from: `www.tandfonline.com`.

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

Ropke, S. and Pisinger, D. (2006a). An adaptive large neighborhood search heuristic for the pickup and delivery problem with time windows. *Transportation Science*, 40(4):455–472.

Ropke, S. and Pisinger, D. (2006b). A unified heuristic for a large class of vehicle routing problems with backhauls. *European Journal of Operational Research*, 171(3):750–775.

Shaw, P. (1997). A new local search algorithm providing high quality solutions to vehicle routing problems. *APES Group, Dept of Computer Science, University of Strathclyde, Glasgow, Scotland, UK.*

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

Tompkins, J., White, J., Bozer, Y., and Tanchoco, J. (2003). *Facilities Planning, 3rd ed.* John Wiley & Sons.

Vaughan, T. (1999). The effect of warehouse cross aisles on order picking efficiency. *International Journal of Production Research*, 37(4):881–897.

Weaver, K. A., Baumann, H., Starner, T., Iben, H., and Lawo, M. (2010). An empirical task analysis of warehouse order picking using head-mounted displays. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 1695–1704. ACM.

Whittley, I. and Smith, G. (2004). The attribute based hill climber. *Journal of Mathematical Modelling and Algorithms*, 3(2):167–178.

Aalto-DD 62/2014

ISBN 978-952-60-5674-6
ISBN 978-952-60-5675-3 (pdf)
ISSN-L 1799-4934
ISSN 1799-4934
ISSN 1799-4942 (pdf)