University of Ljubljana
Faculty of computer and information science

Matic Horvat

An Approach to Order Picking Optimization in Warehouses

diploma thesis

undergraduate university study programme
computer and information science

Mentor: Acad. Prof. Ivan Bratko, PhD

Ljubljana 2012

Univerza v Ljubljani
Fakulteta za računalništvo in informatiko

Matic Horvat

Pristop k optimizaciji zbiranja blaga za odpremo v skladiščih

diplomsko delo

univerzitetni študijski program prve stopnje
računalništvo in informatika

Mentor: akad. prof. dr. Ivan Bratko

Ljubljana 2012

Besedilo je oblikovano z urejevalnikom besedil LaTeX.

University
of Ljubljana

Faculty of Computer
and Information Science

Tržaška 25
1000 Ljubljana, Slovenia
Phone: +386 1 476 84 11
Fax: +386 1 426 46 47
www.fri.uni-lj.si
e-mail: dekanat@fri.uni-lj.si

No. of dissertation: 00025/2012
Date: 11.04.2012

University of Ljubljana, Faculty of Computer and Information Science issues the following dissertation:

Candidate: **MATIC HORVAT**

Title: **AN APPROACH TO ORDER PICKING OPTIMIZATION IN WAREHOUSES**

Type of dissertation: Undergraduate dissertation of first cycle

Topic of the thesis:

The planning of order picking is an important function in warehouse management. Time optimisation of order picking is a combinatorially complex optimisation problem. This problem is specially difficult in the case of "randomised storage" where the same type of product is stored at multiple locations in the warehouse. Your task is to develop an approach to the optimisation of order picking for randomised storage. Experimantally test and evaluate your approach on a realistic type warehouse.

Mentor:

Acad. Prof. Ivan Bratko, PhD

Fakulteta za računalništvo in informatiko
2
UNIVERZA V LJUBLJANI

Dean:

Prof. Nikolaj Zimic, PhD

Univerza v Ljubljani
Fakulteta za računalništvo in informatiko

Tržaška 25
1001 Ljubljana, Slovenija
telefon: 01 476 84 11
01 476 83 87
faks: 01 426 46 47
01 476 87 11
www.fri.uni-lj.si
e-mail: dekanat@fri.uni-lj.si

Št. naloge: 00025/2012
Datum: 11.04.2012

Univerza v Ljubljani, Fakulteta za računalništvo in informatiko izdaja naslednjo nalogo:

Kandidat: **MATIC HORVAT**

Naslov: **PRISTOP K OPTIMIZACIJI ZBIRANJA BLAGA ZA ODPREMO V SKLADIŠČIH**

AN APPROACH TO ORDER PICKING OPTIMIZATION IN WAREHOUSES

Vrsta naloge: Diplomsko delo univerzitetnega študija prve stopnje

Tematika naloge:

Pomembna funkcija v vodenju skladišča je planiranje nabiranje blaga za posamezna naročila za odpremo blaga iz skladišča. Časovna optimizacija nabiranja blaga je kombinatorično zahteven optimizacijski problem. Ta je še posebej težak v primeru t.i. »neurejenih« skladišč, kjer blago v skladišču ni sistematično urejeno po vrstah izdelkov. Naloga je razviti pristop k optimizaciji nabiranja blaga v neurejenih skladiščih. Pristop naj bo eksperimentalno preizkušen in ovrednoten na realnem tipu skladišča.

Mentor:

akad. prof. dr. Ivan Bratko

Fakulteta za računalništvo in informatiko • UNIVERZA V LJUBLJANI • 2

Dekan:

prof. dr. Nikolaj Zimic

Izjava o avtorstvu diplomskega dela

Spodaj podpisani Matic Horvat, z vpisno številko **63090032**, sem avtor diplomskega dela z naslovom:

Pristop k optimizaciji zbiranja blaga za odpremo v skladiščih

S svojim podpisom zagotavljam, da:

- sem diplomsko delo izdelal samostojno pod mentorstvom akad. prof. dr. Ivana Bratka,
- so elektronska oblika diplomskega dela, naslov (slov., angl.), povzetek (slov., angl.) ter ključne besede (slov., angl.) identični s tiskano obliko diplomskega dela
- soglašam z javno objavo elektronske oblike diplomskega dela v zbirki "Dela FRI".

V Ljubljani, dne 11. julija 2012 Podpis avtorja:

Za mentorstvo, vodenje in pomoč se zahvaljujem dr. Ivanu Bratku.

Posebej se želim zahvaliti dr. Mateju Guidu, ki me je, skorajda dobesedno, naučil pisati.

Iskrena zahvala gre dr. Aleksandru Sadikovu, ki je bil tudi ob nemogočih urah pripravljen prediskutirati mnoge zagate. Pogovori ob tabli bodo ostali najljubši del izdelovanja diplomske naloge.

Za razvoj in neprestano izboljševanje programske opreme za simuliranje skladišč ter za potrpežljivost pri mnogih vprašanjih, ko se je zataknilo pri programiranju, se zahvaljujem Nejcu Škofiču.

Zahvaljujem se tudi podjetju 3R.TIM in Dejanu Reichmannu za odpiranje okna v svet logistike.

Na koncu se želim zahvaliti vsem domačim za podporo in razumevanje skozi študijska leta. Hvala.

Contents

List of Figures

Razširjeni povzetek

Pobiranje naročil je ena izmed vsakodnevnih operacij v skladiščih, katere namen je zbiranje produktov iz skladiščnih lokacij za izpolnitev naročil strank. Po mnogih ocenah operacija pobiranja naročil v povprečju predstavlja več kot 50% vseh operativnih stroškov vodenja skladišča. Glavni vzrok za to je pogosto zaposlovanje človeških pobiralcev, saj avtomatiziranje pobiranja naročil zahteva velike investicije. Pobiranje naročil v zadnjih letih med stroko zato postaja vse bolj privlačno področje za izboljševanje produktivnosti v skladiščih.

Nizkonivojski sistemi pobiralec-k-produktom predstavljajo klasični način pobiranja naročil v skladiščih. Pobiralec naročil se namreč sprehaja ali vozi po skladišču s pobiralno napravo, ki je pogosto neke vrste voziček ali vozilo s prostorom za shranjevanje produktov, ter na njej zbira produkte, ki jih je potrebno pobrati, da izpolni naročila strank. Tak tip pobiralnih sistemov predstavlja več kot 80% vseh pobiralnih sistemov v Zahodni Evropi. Kljub njihovi razširjenosti pa so pogosto zapostavljeni s strani akademskih raziskovalcev, ki večji poudarek dajejo optimizaciji drugih tipov pobiralnih sistemov. V tem diplomskem delu se osredotočimo na optimizacijo v nizkonivojskih sistemih pobiralec-k-produktom.

Optimizacija pobiranja naročil v skladiščih po ugotovitvah raziskovalcev omogoča pomembne prihranke pri stroških dela in zaradi krajših časov dostave izboljša kvaliteto storitev za stranke. Pri optimizaciji se osredotočamo na zmanjšanje celotnega časa potrebnega za pobiranje naročil. Delovni čas pobiralcev naročil je sestavljen iz večih podnalog, ki so: premikanje po

skladišču, iskanje produkta, pobiranje produkta in priprava. Čas, ki ga pobiralec porabi za iskanje in pobiranja, se pogosto smatra za konstanten, medtem ko je čas priprave zanemarljiv. Na drugi strani čas premikanja predstavlja 50% celotnega delovnega časa pobiralca naročil. Če predpostavimo, da se pobiralci gibljejo s konstantno hitrostjo, cilj optimizacije postane celotna prepotovana razdalja, ki jo pobiralci naročil prepotujejo, da izpolnijo vsa naročila strank.

K optimizaciji pobiranja naročil je mogoče pristopiti s spreminjanjem večih medsebojno odvisnih taktik. Med pogosto optimiziranimi taktikami so: taktika uskladiščevanja, taktika združevanja naročil in taktika usmerjanja pobiralcev naročil. Pri načrtovanju novega skladišča je pomembna tudi optimizacija ureditve skladišča. S slednjo se v tem delu ne bomo ukvarjali.

Taktika uskladiščevanja ureja shranjevanje prispelih produktov v skladiščne lokacije. Namensko uskladiščevanje ohranja eno ali več sosednjih skladiščnih lokacij za shranjevanje enega tipa produkta. Na drugi strani naključno uskladiščevanje vsakemu prihajajočemu tipu produkta dodeli naključno prazno skladiščno lokacijo.

Taktika združevanja naročil ureja preoblikovanje naročil strank v seznam pobiranja. Seznam pobiranja je seznam produktov, ki jih pobiralec zbere v enem obhodu po skladišču, preden se vrne na izhodiščno mesto in jih zloži s pobiralne naprave. Pobiranje produktov enega naročila stranke v večih obhodih po skladišču pogosto ni dovoljeno, saj povzroči dodatne nesprejemljive stroške sortiranja. Pri taktiki posameznega pobiranja naročil pobiralec hkrati zbira produkte le enega naročila stranke. Pri taktiki množičnega pobiranja naročil pobiralec hkrati zbira produkte večih naročil strank. Slednji način pobiranja naročil je mogoč, ko je velikost naročil strank majhna v primerjavi s kapaciteto pobiralne naprave. Množično pobiranje naročil povzroči nov optimizacijski problem razvrščanja naročil v skupine, ki jih pobiralci obravnavajo hkrati. Ker gre za težak problem, se ga pogosto rešuje s pomočjo hevrističnih metod. Te lahko razdelimo v tri glavne pristope: algoritme osnovane na prednostnih pravilih, semenske algoritme in algoritme s prihranki. Nekateri

raziskovalci so se lotili reševanja tega problema z uporabo metahevristik, kot sta lokalno preiskovanje in populacijske metode.

Taktika usmerjanja pobiralcev ureja vrstni red, v katerem pobiralci pobirajo produkte s seznama pobiranja, na način, da je prepotovana razdalja čim krajša. Pri individualnem usmerjanju je vsak seznam pobiranja optimiziran posamezno. Za določene ureditve skladišč obstajajo optimalni algoritmi za rešitev tega problema. Pogosteje se za reševanje uporabljajo hevristike, saj so optimalni algoritmi neprilagodljivi različnim ureditvam skladišč ter pogosto uredijo seznam pobiranja na način, ki se pobiralcem zdi kompleksen in nelogičen. Med hevristikami se najpogosteje uporablja standardizirano usmerjanje z uporabo usmerjevalnih strategij. Mednje sodijo: vrnitvena strategija, strategija S-oblike in strategija največje vrzeli. Eksperimentalne raziskave so pokazale, da se strategija največje vrzeli najbolj približa optimalni rešitvi (v povprečju je 9-10% slabša). Ta strategija maksimizira neprehojeno razdaljo v prehodih skladišča.

Naštete taktike so medsebojno odvisne in le njihova hkratna rešitev bi vodila k globalno optimalni rešitvi problema optimizacije pobiranja naročil. Vendar njihova hkratna optimizacija v enem modelu predstavlja neobvladljiv problem. Običajno se raziskovalci osredotočijo na eno ali dve izmed naštetih taktik hkrati, medtem ko se v praksi o taktikah odloča zaporedno.

V tem diplomskem delu smo raziskali medsebojni vpliv taktike uskladiščevanja, specifično različice naključnega uskladiščevanja, in treh pristopov k reševanju problema razvrščanja naročil v skupine. Vendar je naključno uskladiščevanje vpeljalo nov problem izbire produkta, saj je produkt istega tipa shranjen na večih lokacijah po skladišču. Zato smo pripravili običajen pristop k optimizaciji pobiranja naročil in ga razširili z novim korakom izbire produktov, ki smo ga rešili z uporabo algoritma RTA*.

Celoten pristop optimizacije pobiranja naročil je sestavljen iz večih zaporednih korakov. Sprva z uporabo algoritma RTA* izberemo specifične produkte v skladišču, ki jih bodo pobiralci zbrali. Nato razvrstimo naročila strank v skupine, ki jih pobiralci naročil obravnavajo hkrati. Na koncu z

uporabo strategije največje vrzeli določimo vrstni red pobiranja produktov znotraj ene skupine naročil strank.

Za rešitev problema izbire produkta smo uporabili hevristični preiskovalni algoritem RTA*. Ker preiskovalni prostor raste eksponentno z velikostjo skladišča in naročil, smo dodatno razvili metodo za zmanjšanje preiskovalnega prostora z uporabo hierarhičnega razvrščanja produktov. Problem izbire produkta je v praksi najpogosteje rešen z uporabo principa FIFO, kjer je izbran produkt, ki je najdlje v skladišču. Izbira produktov po drugih principih lahko privede do staranja inventarja skladišča in slabše porabe prostora. Čeprav se s tem problem nismo neposredno ukvarjali v tem diplomskem delu, pa RTA* omogoča uporabo kombinirane hevristike, ki bi upoštevala tako bližino produktov kot tudi njihovo starost.

Za rešitev problema razvrščanja naročil v skupine smo uporabili tri algoritme kot predstavnike treh pristopov k reševanju tega problema. Algoritem FCFS naročila razvršča v skupine glede na vrstni red, v katerem so bila narejena, kar najpogosteje pomeni naključni vrstni red. Ta algoritem se pogosto uporablja kot osnovni algoritem, s katerim se primerjajo naprednejši pristopi. Dodatno smo uporabili dva konkurenčna algoritma, semenski algoritem in algoritem s prihranki, ki temeljita na naprednejših hevristikah, ki upoštevajo razdalje med produkti v skladišču.

Uporaba taktike namenskega uskladiščevanja povzroči popolnoma urejeno skladišče, saj se vsak tip produkta nahaja na eni sami lokaciji v skladišču ali na večih sosednih lokacijah. Uporaba taktike naključnega uskladiščevanja povzroči popolnoma neurejeno skladišče, saj se produkti istega tipa nahajajo v večih skupinah po vsem skladišču. Taktike uskladiščevanja nismo modelirali eksplicitno, saj bi to zahtevalo simuliranje prihajanja novih produktov v skladišče. Namesto tega smo oblikovali algoritem, ki prazno skladišče napolni z izdelki po vzoru naključnega in namenskega uskladiščevanja. Algoritem namreč s parametrom omogoča upravljanje urejenosti skladišča, ki se razteza od popolnoma urejenega do popolnoma neurejenega.

Naš pristop smo ovrednotili z eksperimenti v simuliranem okolju skladišča.

Eksperimente smo izvedli v treh skladiščih različnih velikosti. Za vsako izmed skladišč smo naključno zgradili 50 množic naročil strank, katere smo uporabili za ocenjevanje različnih vidikov našega pristopa k optimizaciji pobiranja naročil. V prvi vrsti smo želeli ugotoviti odnos med urejenostjo skladišča in skupno dolžino obhodov po skladiščih. Dodatno nas je zanimala izvedljivost našega pristopa pri naraščujoči velikosti skladišč in števila naročil.

Rezultati eksperimentov so pokazali, da ima nizek nivo urejenosti skladišča za posledico tudi do 14% zmanjšanje dolžin obhodov pobiralcev po skladišču v primerjavi z visokim nivojem urejenosti. Tako zmanjšanje je primerljivo z zmanjšanjem doseženim z uporabo semenskega algoritma namesto osnovnega algoritma FCFS. Eksperimenti so prav tako pokazali, da se naš pristop dobro obnese pri velikih skladiščih in večjem številu naročil.

Naše rešitve problema izbire produkta na žalost nismo mogli eksperimentalno primerjati z drugimi pristopi iz literature, saj je bil predlagan le en pristop reševanja tega problema, ki ni združljiv z ostalimi deli našega pristopa. Na drugi strani bi bila primerjava z de facto pristopom FIFO nepravična, saj pristopa optimizirata dva različna cilja: dolžino prehojene poti in starost skladišča. Taka primerjava zahteva simulacijo skladiščnih operacij v daljšem časovnem obdobju in merjenje potencialnega staranja inventorija skladišča. V tem diplomskem delu se takih simulacij nismo lotili, a predstavljajo zanimivo področje za nadaljne delo.

Ključne besede

Skladišče, Pobiranje naročil, Optimizacija

Abstract

The conventional approach to order picking optimization divides the process in three main decision areas: storage policy, order consolidation policy, and routing policy. In practice, we encountered variants of randomized storage in warehouses that range from completely ordered to completely unordered. We investigated the effect of the order level of the warehouse on the picking tour length. An additional problem encountered in warehouses with randomized storage that is not addressed by the conventional methods of order picking optimization is item selection. The item selection problem is concerned with selecting a single product to pick from multiple products throughout the warehouse so as to satisfy a customer order. We extended the conventional approach to order picking optimization by proposing a solution to the item selection problem using RTA* algorithm. Alongside established solutions to the order batching and picker routing problems, we integrated this into a complete sequential approach to order picking optimization. We evaluated our approach by experiments in simulated warehouse environment. We demonstrated that low order level of the warehouse is to be preferred compared to high order level as it results in shorter picking tours. Additionally, we have shown that our approach scales well with increasing warehouse size and number of orders.

Keywords

Warehouse, Order picking, Optimization

Chapter 1

Introduction

Order picking is a warehousing operation that deals with picking products from storage locations in order to satisfy customer orders. It has been estimated that order picking accounts for up to 50% of the total warehouse operating costs [1]. This is in large part due to the fact that order picking still often requires involvement of human order pickers, as automating order picking systems necessitates large investments [2, 3]. Because of this, order picking has in recent years become an area of increased interest among warehouse professionals for improving productivity in warehouses [2].

The conventional approach to order picking optimization divides the process in three main decision areas: storage policy, order consolidation policy, and routing policy. Although they are strongly interdependent and only a simultaneous solution to all of them could lead to a global optimal solution to order picking optimization problem [4], inclusion of all decisions in one model is never done in practice because it is computationally intractable [2]. Instead, researchers focus on one or two of these decision areas simultaneously, while in practice decisions are made sequentially [2].

In this thesis, we investigate the interaction between storage policy, specifically a variant of randomized storage, and three different approaches to solving order batching problem with a fixed routing strategy. However, randomized storage introduces a new problem as the same type of product is

stored at multiple locations in the warehouse. A specific product to be picked can be chosen from any of these locations. We refer to this as the *item selection problem* and propose a sequential solution using the RTA* algorithm. We implement the optimization approach and evaluate it with simulation experiments.

We demonstrate that low order level of the warehouse results in up to 14% shorter picking tours compared to high order level. This is a significant reduction in picking tour lengths, which, surprisingly, hasn't been discussed in the research literature. The second main contribution of this thesis is highlighting the problem of item selection, which has been a subject of a single optimization attempt so far, and proposing a solution to solving it using heuristic search.

The remainder of this thesis is organized as follows. In the next chapter we introduce the domain of warehouses and order picking, and highlight the optimization problem we are trying to solve. In Chapter 3, we explain our approach to optimizing the order picking. In Chapter 4, we give an overview of the experiments used to evaluate our approach. The results of the experiments are presented in Chapter 5. We discuss the implications of the results in Chapter 6 and conclude the thesis with Chapter 7.

Chapter 2

Order picking and existing solutions

"Warehousing is an integral part of every logistics system" [5, p266]. It forms a link between producers and customers to ensure constant and timely delivery of products. The main purpose of warehousing is storage and buffering of products, ranging from raw materials and parts, to finished goods. Warehousing can be used at any point in the supply chain: at producer (e.g. factory warehouse), at customer (e.g. shopping centre warehouse), and at any point in between [5].

The term 'warehouse' therefore refers to a *facility* where the main function is storage and buffering of products. Instead of 'warehouse' other terms are sometimes used. While they are similar they can not be used interchangeably when defining their main functions. We use the term 'distribution center' (DC) when the emphasis is moved from storage to distribution of predominantly high-demand products. When storage of products is of lesser or no importance the terms 'transshipment', 'cross-dock', and 'platform' center are used. In these facilities products are rarely stored for a long period of time and are usually moved directly from receiving to shipping dock [5, 2].

As our focus is order picking from storage, we will use the term 'warehouse' throughout the thesis.

The remainder of this chapter is organized as follows. In Section 2.1 we briefly describe the main warehousing operations. In Section 2.2 we focus on order picking operation. We continue with the description of order picking systems in Section 2.3, and finish the chapter with Section 2.4 dealing with order picking optimization.

2.1 Warehousing operations

Warehousing operations can be divided into several functions [5, 2], mainly:

- *Receiving*, which consists of unloading of products from transportation vehicles to receiving docks, inspection of products for deficiencies or missing products, and updating warehouse inventory records to reflect changes,
- *Transfer* or *put away*, which includes moving products from receiving dock to assigned storage locations, shipping dock or other areas in the warehouse, and moving products between these areas,
- *Order picking*, which consists of collecting required quantities of specified products from storage locations in order to satisfy customer orders,
- *Shipping*, which includes loading products onto transportation vehicles, inspection of products to be shipped, and updating warehouse inventory records. It can additionally include sorting and packaging of products.

2.2 Order picking

Order picking is based on customer orders. A customer order consists of order lines, each line specifying the product and required quantity. Order picking is therefore a warehouse operation that satisfies customer orders by picking the required products from storage locations and bringing them to an

area dedicated for collecting the assembled customer orders, usually referred to as a depot [6, 3].

It has been estimated that order picking accounts for up to 50% of the total warehouse operating costs [1]. This is in large part due to the fact that order picking often still requires the involvement of human order pickers, as automating order picking systems necessitates large investments [2, 3]. Because of this, order picking has in recent years become an area of increased interest among warehouse professionals for improving productivity in warehouses [2].

2.3 Order picking systems

Order picking can be performed in a variety of high-level approaches often referred to as *order picking systems* (OPS). According to Dallari et al. [7] their design depends on several warehousing elements, ranging from products (e.g. number, size, value), customer orders (e.g. number, size), to design and layout of warehouse areas.

In this thesis, we use the classification of order picking systems proposed by Dallari et al. [7]. Their classification is based on four decisions made when designing the order picking systems: (1) who will pick products from storage locations (i.e. humans or machines), (2) who will move in the picking area (i.e. pickers or products), (3) are conveyors used to transport picked products, and (4) what is the picking strategy. The resulting classification of order picking systems is shown in Figure 2.1.

We will describe the four OPS in order of increasing level of automation while omitting the description of completely automated picking as it is rarely used in practice.

Picker-to-parts system involves human pickers that move along the aisles picking products on foot using carts or riding on specialized vehicles. Pickers can pick a single order at a time or a batch of multiple orders. These OPS are further divided in *low-level* picker-to-parts systems, in which prod-

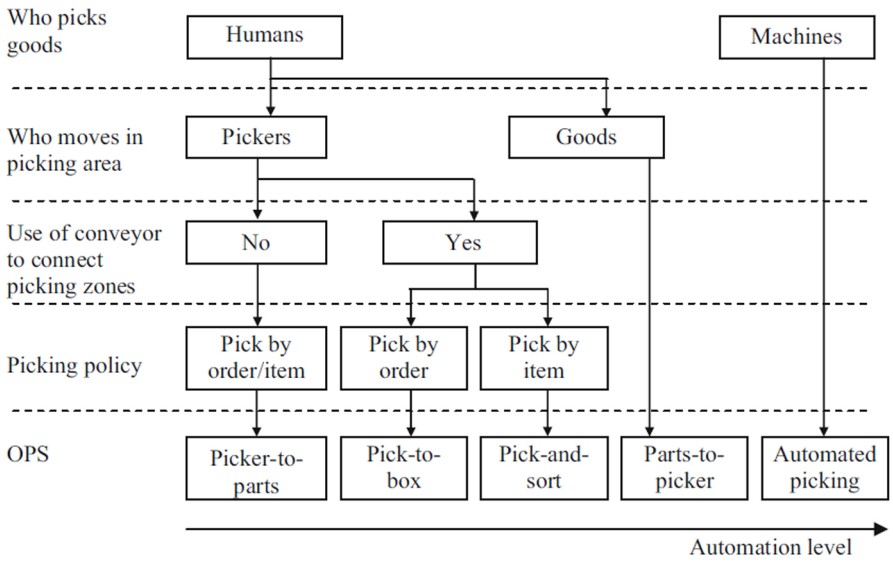

Figure 2.1: Classification of order picking systems (taken from Dallari et al. [7]).

ucts are directly accessible from warehouse floor, and *high-level* picker-to-parts systems (also referred to as man-on-board systems), where a lifting truck or a crane is needed to pick products from high storage racks [7].

Pick-to-box system divides the picking area in multiple zones each assigned to one or more pickers. Each picker only picks the part of the order that is in his assigned zone. An order can be picked sequentially, zone after zone, or simultaneously in all zones. The picking area zones can be connected by a conveyor belt, which carries the boxes of partially or fully completed orders. The main benefits of pick-to-box systems compared to picker-to-parts systems are the reduction of travel time of pickers as they are confined to a smaller area, and the resulting reduction of traffic congestion. It also enables pickers to become familiar with their area and therefore reducing search time and consequently enabling faster retrieval of products from storage locations [2, 7].

Pick-and-sort system no longer requires an order to be picked in its

entirety in one picker tour. Instead, products are picked independently of orders and placed on a conveyor which takes them to a sorting area, where an automated sorter assembles individual products into customer orders. As the picking locations are visited less frequently, this results in a reduction in travel time and increase in productivity. However, implementing an automated sorter requires a large investment [7].

Parts-to-picker system uses automated devices to retrieve bins or pallets from storage aisles to pick locations, where pickers select the required number of products and the rest are returned to storage by an automated device. Types of automated devices include carousels, modular vertical lift modules, mini-loads, and automated storage and retrieval systems (AS/RS). As there is no need for pickers to move through picking area, use of such systems results in reduced order picking time [2, 7]

2.4 Order picking optimization

In this thesis, we concentrate on optimization of *low-level picker-to-parts systems*. Picker-to-parts systems are the most commonly encountered OPS in practice, with low-level order picking systems representing more than 80% of all order picking systems in Western Europe. Despite their prevalence, they are often neglected by academic researchers, who put more emphasis on optimization of high-level picker-to-parts systems and AS/RS [2].

Optimizing order picking is reported to result in significant savings in labour costs. In the short term, it reduces the need for overtime and employment of temporary staff, while in the long term, it provides an option for decreasing the number of permanent staff. Additionally, customer service is improved due to shorter delivery times [3, 4].

Before proceeding with discussion of order picking optimization, we need to clarify some common terms used in low-level picker-to-parts systems:

- A warehouse employee that is tasked with order picking is referred to as *order picker* or *picker*.

- The order picker uses a *picking device* to collect products. The picking devices vary between warehouses, but are usually a form of a cart or a motorized vehicle with storage capacity.

- A *pick list* is an ordered list of products to be picked in specified quantities. A pick list should not be confused with a customer order. The difference is that a pick list contains products with assigned storage locations in the warehouse, whereas customer order does not.

- A *picking tour* is the path that the order picker traverses to fulfil a single pick list. It starts at the depot with an empty picking device. The products are picked and placed on the picking device according to the pick list order. When the pick list is completed, the order picker returns to the depot to unload the picking device. The number of products that can be picked in a single picking tour and therefore the number of products specified on the pick list is limited by the capacity of the picking device.

In low-level picker-to-parts systems splitting of customer order between multiple pick lists is usually not permitted, since "it would result in an additional, non-acceptable sorting effort" [3].

Order picking optimization focuses on reducing the total order picking time [4]. The time of a single picking tour is comprised of [8, 3]:

- *Travel time* between pick locations, and between pick locations and the depot,

- *Search time* for products after arriving at the pick location,

- *Pick time* for picking products from storage locations and placing them on a picking device,

- *Set-up time* for administrative tasks (e.g. inventory updating), and emptying of picking device at the depot.

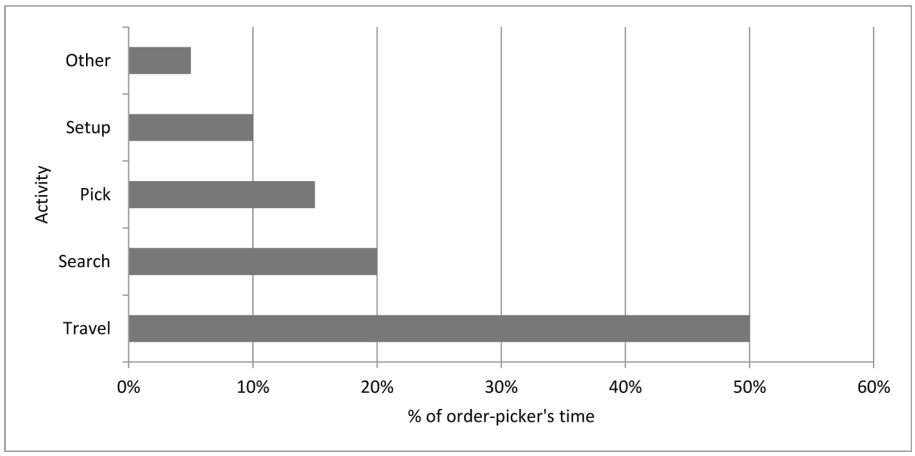

Figure 2.2: [9] Typical order picker's time distribution.

Figure 2.2 shows the typical distribution of order picker's time.

Travelling is the most time consuming subtask of order picking. Pick and search times are constant regardless of picking tour composition, while set-up time is most often neglectable [3]. Therefore, we are left with travel time as the optimization objective. However, we can assume that order pickers travel at constant speed, which results in optimization of total travel distance [10, 3].

There are several interdependent policies that need to be considered when optimizing order picking. According to Wäscher [4] these are primarily: storage policy, order consolidation policy, and routing policy. When designing a new warehouse, layout optimization also presents an important aspect of optimizing order picking. However, as we focus on order picking optimization in existing warehouses, we will not discuss layout optimization. Figure 2.3 shows an overview of order picking approaches.

2.4.1 Storage policy

Storage policy is concerned with assigning products to storage locations when new products arrive in the warehouse.

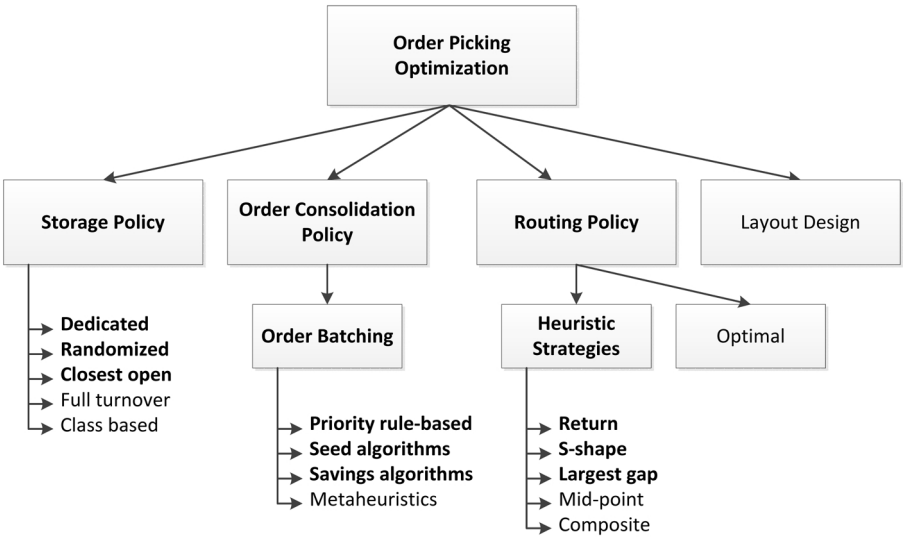

Figure 2.3: Order picking optimization approaches. The methods and policies displayed in bold are presented in the following sections.

Dedicated storage policy maintains a single storage location (or a number of adjacent locations) for each product type over a relatively long period of time [4]. Dedicated storage policy helps pickers become familiar with warehouse inventory locations and can over time result in reductions of order picking time [2, 4]. However, the downside is the low warehouse space utilization as storage locations are reserved even though they are not currently used [2].

Randomized storage policy assigns a random empty location to every type of incoming product. This results in products of the same type being spread throughout the warehouse [4]. Randomized storage policy has a high space utilization [2]. However, in de Koster et al. [2] the authors argue that randomized storage is only possible in a computer-guided warehouse.

Closest open location storage policy is derived from randomized storage if the order pickers are able to choose the empty location themselves [2].

2.4.2 Order consolidation policy

Order consolidation policy is concerned with transforming customer orders to unordered picking lists.

With *single order picking* policy a customer order is directly transformed into a picking list, which results in one order being picked at a time.

With *order batching* policy multiple orders forming a batch are picked at the same time. Order batching is possible when order size is small compared to the capacity of picking device. Order batching gives rise to *order batching problem*, i.e. what orders should be picked at the same time so as to minimize total order picking distance [4]. The problem was proven to be NP-hard if the batches consist of more than two orders, which means it is only possible to solve it optimally for small problem instances [10]. Usually the problem is solved using heuristic methods. There exist three conventional approaches to solving the order batching problem. We describe these approaches below. Additionally, metaheuristics, such as local search, tabu search, and population based approaches have been used to solve the order batching problem.

Priority Rule-Based algorithms consist of two phases. In the first phase, priority is assigned to every customer order. In the second phase, customer orders are sequentially assigned to batches in the decreasing order of priority while ensuring the capacity of the batch does not exceed the capacity of the picking device. A specific batch for a customer order can be chosen according to Next-Fit, First-Fit, or Best-Fit rule. The priority of a customer order can be determined in a variety of approaches. However, the most straightforward one is the First-Come-First-Served (FCFS) rule, which assigns higher priority to the customer orders that arrived earlier. In most situations, this results in a random sequence of customer orders. However, the results obtained with FCFS rule are commonly used as a baseline solution with which other order batching heuristics are compared with [3].

Seed algorithms construct batches sequentially. A batch is constructed in two phases. In the first phase, a seed order is chosen according to the seed

selection rule. In the second phase, additional orders are added to the batch according to the accompanying-order selection rule until there are no orders left that can be added to the batch without violating the picking device capacity constraints. Batches are constructed until all customer orders have been assigned to a batch [3].

Accompanying-order selection rules are usually based on the 'distance' from the seed [2]. In the second phase, we distinguish two ways of determining the seed: the seed is the first customer order that was assigned to the batch (single mode), or the seed consists of all the customer orders that were assigned to the batch so far (cumulative mode) [3].

There exist a large number of seed selection and accompanying-order selection rules. Some examples of seed selection rule are: random customer order, smallest number of picking locations, greatest number of picking locations, and smallest number of picking aisles [11]. Some examples of accompanying-order selection rule are: random customer order, smallest number of additional picking locations, smallest number of additional picking aisles, and greatest number of identical picking locations [11].

Savings algorithms are based on the well-known Clarke and Wright algorithm for the vehicle routing problem [12, 4]. The central idea of the savings algorithms is a saving, i.e. the reduction in distance travelled if two orders or batches are picked in the same tour compared to the combined distance of individual tours. Customer orders are added to batches according to the largest saving without violating the capacity constraint. The savings are recalculated every time a customer order is added to a batch or two batches are joined together [3].

2.4.3 Routing policy

Routing policy is concerned with ordering a pick list in a sequence that will minimize the travel distance of an order picker.

With *individual routing* policy each pick list is optimized individually. This optimization is a variant of the well-known Travelling Salesman Prob-

lem (TSP) [13]. Such problems are usually solved using approximation and heuristic algorithms which provide suboptimal solutions. However, for some warehouse layout configurations optimal polynomial-time algorithms exist, most notably the algorithm proposed in Ratliff and Rosenthal [13].

In practice, optimal and heuristic approaches for individually optimizing the length of a picking tour are not often used [3], due to the following reasons. Firstly, the optimal algorithm presented in Ratliff and Rosenthal [13] is complex and not easily adaptable to different warehouse layouts [8]. Secondly, the sequences produced with such algorithms are often not straightforward and seem illogical to order pickers, who then as a result deviate from the calculated routes [8, 10], defeating the purpose of optimizing picking tours. Thirdly, aisle congestion is not accounted for in a standard optimal routing algorithm, while it is possible to avoid it using specific heuristic methods [2].

Instead, *standardized routing* with various *routing strategies* is often used in practice [4]. The most well known routing strategies include return strategy, S-shape strategy, and largest gap strategy. An example of the aforementioned strategies is shown in Figure 2.4.

When using a *return strategy* the order picker visits an aisle if there are pick locations in that aisle. He enters it from the front cross aisle and picks the products from the pick list in that aisle. When he picks the most distant product, he returns to the front cross aisle.

In *S-shape* strategy the order picker traverses an entire aisle that contains at least one pick location. This results in the order picker alternating between entering aisles from front and back cross aisle.

Largest gap strategy tries to maximize the distance in the aisle that is not traversed. A gap is the distance between the start of an aisle and the first pick location, two adjacent pick locations, and last pick location and end of the aisle. The order picker returns to the start of the aisle at the pick location that is part of the largest gap. If there are pick locations yet to be visited in this aisle, he will visit them by entering the aisle from back cross

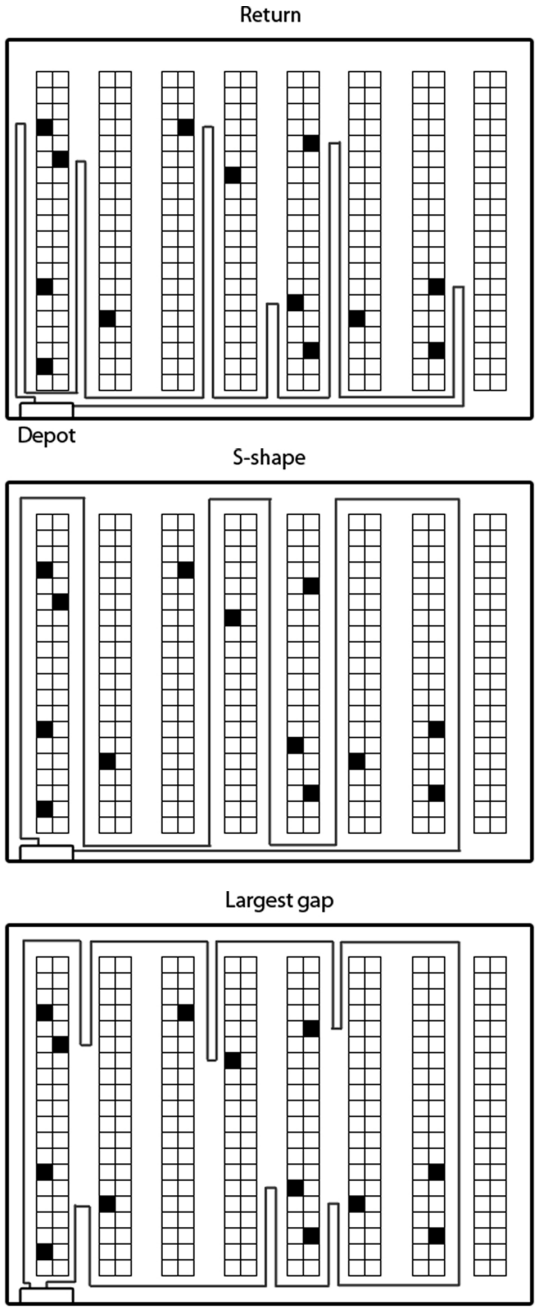

Figure 2.4: An example of a picking tour constructed using different routing strategies. Black squares represent products that need to be picked (pick locations).

aisle later in the pick tour.

Petersen [14] compared six routing strategies, including return, S-shape, and largest gap, with optimal routing algorithm at different warehouse layouts. The results showed that the best heuristic routing strategies, largest gap and composite, produced on average 9 to 10 percent longer picking tours compared to the optimal algorithm.

Chapter 3

Our approach

The conventional approach to optimizing order picking in low-level picker-to-parts systems in existing warehouses has been described in Section 2.4. It divides the process of order picking optimization in three main decision areas: storage policy, order consolidation policy, and routing policy. Although they are strongly interdependent and only a simultaneous solution to all of them could lead to a global optimal solution to order picking optimization problem [4], inclusion of all decisions in one model is not done in practice because it is intractable [2]. Instead, researchers focus on one or two of these decision areas simultaneously, while in practice decisions are made sequentially [2].

In this thesis, we investigate the interaction between storage policy, specifically a variant of randomized storage, and three different approaches to solving order batching problem with a fixed routing strategy. However, randomized storage introduces a new problem as the same type of product is stored at multiple locations in the warehouse. A specific product to be picked must be chosen from any of these locations. We refer to this as the *item selection problem*. We propose a sequential solution for solving it.

The remainder of this chapter is organized as follows. We first provide a general overview of our solution approach in Section 3.1. In Section 3.2 we present our implementation of three algorithms used to solve the order batching problem. In Section 3.3 we describe the item selection problem, and

our approach to solving it. We propose an attempt at improving the existing picking tours with item reselection in Section 3.4. Finally, in Section 3.5 we describe the implementation of storage policy.

3.1 Overview of our approach

The proposed approach to order picking optimization consists of several sequential steps. First, we use the RTA* algorithm to assign specific products to customer orders. Then the customer orders are batched using one of the order batching algorithms. The result of order batching is a number of picking tours. Optionally, item reselection is applied after the order batching. Finally, picker routing orders the picking tours according to a routing policy.

For the routing policy we implemented the largest gap strategy. We base our decision on the comparison made by Petersen [14] that was discussed in the previous chapter. The rest of the steps in our approach are explained in detail in the following sections.

3.2 Order batching

In the previous chapter, we described the order batching problem as choosing what orders should be picked at the same time to minimize total order picking distance [4]. We briefly described three conventional approaches to solving it: priority rule-based algorithms, seed algorithms, and savings algorithms. For our approach to the order picking optimization we implemented a representative algorithm for each of the three approaches using specific heuristics. In the following subsections, we present each of the three algorithms we implemented in more detail, including the heuristics used where applicable.

3.2.1 First-Come-First-Served (FCFS) algorithm

The First-Come-First-Served algorithm is an example of the priority rule-based algorithms. It assigns the customer orders a priority based on the order they arrived in. In most situations this results in a random sequence of customer orders. However, the results obtained with FCFS rule are commonly used as a baseline solution which other order batching heuristics are compared with [3]. For this reason we present it here in greater detail.

The pseudocode of the algorithm is shown in Figure 3.1. The algorithm expects the list of customer orders sorted by the time of arrival as its input. In our model, all customer orders are made at the same time, therefore the ordering is random.

```
Input: order_list: list of customer orders ordered by time of arrival
Output: batch_list: list of batches
initialize batch_list: empty list of batches;
for order in order_list do
    assigned = false;
    for batch in batch_list do
        if batch size + order size <= picking device capacity then
            add order to the batch;
            assigned = true;
            break;
        end
    end
    if not assigned then
        create a new batch and add it to batch_list;
        add order to the batch;
    end
end
```

Figure 3.1: FCFS algorithm pseudocode

In the algorithm, orders are sequentially assigned to batches according to the First-Fit rule, i.e. to the first batch (according to the order they were opened) that provides free sufficient capacity not to violate the picking device capacity constraint. If no such batch exists, a new batch is opened and the order is assigned to it.

3.2.2 Seed algorithm

We discussed general properties of seed algorithms in the previous chapter. Here we present our implementation of the algorithm using competitive heuristics in greater detail. The algorithm pseudocode is shown in Figure 3.2.

```
Input: order_list: list of customer orders
Output: batch_list: list of batches
initialize batch_list: empty list of batches;
while order_list is not empty do
    open a new batch;
    select an order from order_list according to seed selection rule;
    add the order to the batch and remove order from order_list;
    while orders can be added to the batch do
        select an order from order_list according to accompanying-order
        selection rule that does not violate capacity constraint;
        add the order to the batch and remove it from order_list;
    end
    add the batch to batch_list;
end
```

Figure 3.2: Seed algorithm pseudocode

In the algorithm, the batches are constructed sequentially until there are no customer orders left to be assigned to a batch. Each batch is initialized with a seed order. The seed order is selected using a seed selection rule. Orders are then added to the new batch according to the accompanying-order

selection rule, until there are no orders that can be added to the batch without violating the picking device capacity constraint. The algorithm works in cumulative mode, i.e. the batch seed is made of all customer orders added to the batch so far.

In the previous chapter, we mentioned a few examples of seed selection and accompanying-order selection heuristics. Ho and Tseng [11] compared the performance and interaction between nine seed selection rules and ten accompanying-order selection rules, majority of which are either aisle-based or location-based rules. They found that the combination of heuristics Smallest Number of Picking Aisles (SNPA) as seed selection rule, and Smallest Number of Additional Picking Aisles (SNAPA) as accompanying-order selection produce the shortest picking tours. For this reason we chose them for our implementation of the seed algorithm. We briefly describe them below. An example of the heuristics is shown in Figure 3.3.

Smallest Number of Picking Aisles is a seed selection rule that chooses a customer order that needs to pick products from as few aisles as possible. As the number of aisles that need to be visited will often produce a tie, we use the order size as the tie breaker. A customer order that visits the smallest number of aisles and contains the largest number of products to be picked is therefore chosen as the seed order.

Smallest Number of Additional Picking Aisles is an accompanying-order selection rule that chooses the order that contains products to be picked in smallest number of aisles that were not yet going to be visited by picking products from the seed. As the algorithm works in cumulative mode, the seed contains all orders currently in the batch. As the rule will often produce a tie, we again use the order size as a tie breaker.

3.2.3 Savings algorithm

The general properties of savings algorithms were discussed in the previous chapter. Here we present our implementation of the algorithm in greater detail.

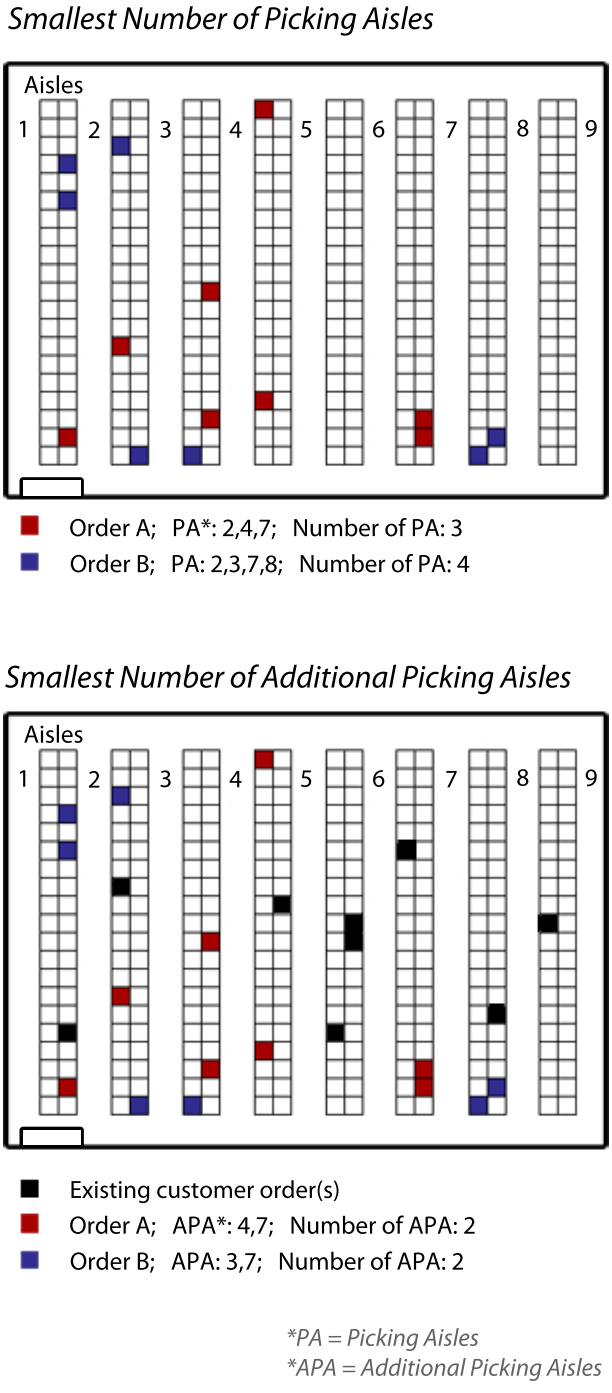

Figure 3.3: An example of the seed selection heuristic SNPA and accompanying-order selection heuristic SNAPA.

The algorithm is based on time savings. A time saving is the reduction in distance travelled if two orders or batches, i and j, are combined in one batch, ij, compared to the sum of distances travelled if orders are picked individually. Therefore, a time saving is:

$$s_{ij} = t_i + t_j - t_{ij} \tag{3.1}$$

The distance travelled is calculated by ordering the picking tour according to the largest gap strategy and computing its distance.

The algorithm pseudocode is shown in Figure 3.4.

Input: order_list: list of customer orders
Output: batch_list: list of batches
initialize batch_list: every customer order forms its own batch;
repeat
    calculate savings for every pair of batches in batch_list;
    create a combined batch of the pair with largest saving that does not violate capacity constraint;
    remove individual batches of the pair from batch_list;
    add the combined batch to the batch_list
until *there were no batches merged in previous iteration*;

Figure 3.4: Savings algorithm pseudocode

In the algorithm, we start by constructing a number of batches each containing one customer order. We then calculate the saving that combining each pair of batches would result in. The combined batch with the largest saving that does not violate the capacity constraint is then constructed and added to the list of current batches, while the individual batches that were joined to form the combined batch are removed from the list. The process is repeated until no batches can be combined due to capacity limit.

3.3 Item selection

Most models of the order picking problem found in the literature today assume that all stock of a particular product is stored at a single storage location in the warehouse [15]. However, this assumption does not hold in practice. Some products might have stock larger than the capacity of a single storage location and must be therefore stored in multiple adjacent storage locations. With randomized storage policy we are investigating in this thesis, the discrepancy is even more apparent as products are intentionally stored at random storage locations throughout the warehouse. Therefore, if a product is available at several storage locations in the warehouse one particular location must be chosen. This is usually solved by selecting products based on First In First Out (FIFO) policy [16], i.e. choosing the product that first arrived in the warehouse.

However, a FIFO policy rarely results in optimal solutions with regard to picking tour length. Van den Berg [16] argues that choosing products based on evaluation other than FIFO is often not acceptable as it causes (1) inventory ageing, and (2) increased storage capacity requirements due to several partially empty pallets of the product stored in the warehouse at the same time. Later in this section, we propose a solution to the item selection problem that enables evaluation of items based both on distances and product age.

To the best of our knowledge there has been only one attempt at solving the item selection problem in the relevant literature. Daniels et al. [15] formulate a model for simultaneously solving the item selection and picker routing problem. They achieve this by generalizing the travelling salesman problem, proving that the new problem formulation is NP-complete, and proposing several extensions of TSP heuristics and a tabu search algorithm for solving it. However, the proposed approach is incompatible with order batching as order batching heuristics rely on product locations for determining which orders to batch together. In the proposed solution the locations of products are unknown until after the picker routing and item selection have

been optimized, therefore preventing the use of order batching techniques. Additionally, routing strategies presented in the previous chapter can not be used with the proposed solution approach.

As order batching enables significant improvements in order picking times compared to picking individual customer orders [6], we propose a sequential solution to the item selection problem using a heuristic search algorithm Real-time A* (RTA*). The sequential solution enables the use of modern approaches to solving the order batching and picker routing problems. As the search space of the RTA* grows exponentially with increasing order and warehouse size, we additionally present a solution to reduce branching factor and consequently the search space using hierarchical clustering.

While not present in our implementation of the RTA* algorithm, the RTA* heuristics can be extended to include the time a product has been in the warehouse as part of the product evaluation. By doing so, we would address one of the concerns over using methods other than FIFO for item selection mentioned by Van den Berg [16] - warehouse ageing. Determining the success of such a heuristic would, however, require simulating the warehouse operations over a long period of time to measure the possible inventory ageing.

3.3.1 The Real-time A* algorithm

We chose the heuristic search algorithm Real-time A* to solve the item selection problem sequentially, i.e. the products to be picked are chosen independently of order batching.

With RTA* algorithm we are able to provide suboptimal solutions. With increasing warehouse and order size it becomes computationally too expensive to calculate the optimal solution in a reasonable time. RTA* is an incomplete search method, meaning that it does not guarantee finding an optimal solution. One of the main benefits it provides is decreased search time which we are also able to control with the lookahead parameter.

Products are selected for one customer order at a time, which means that

the RTA* algorithm is applied to every customer order individually.

The simplified structure of RTA* is outlined in Figure 3.5. A customer order is transformed into a list of products and their respective quantities called the goal list. The current state is the model of the warehouse, including the location of every product and the position of the order picker in the warehouse. The successors of the current state are all models of the warehouse in which the order picker has moved and picked one product.

```
current state s = start_state;
while goal not found do
    Plan: evaluate successors of s by fixed depth lookahead;
    Execute: current state s = successor with minimum backed-up f;
end
```

Figure 3.5: RTA* pseudocode

In the planning stage the algorithm branches on every successor state in which the order picker picks a product that reduces the goal list. The search is recursively conducted until a fixed depth lookahead is reached.

The successor state n is evaluated using formula $f(n) = g(n)+h(n)$, where $g(n)$ is the distance from the current state to n and $h(n)$ is the heuristic evaluation of the node n. If n is at the lookahead horizon it is evaluated statically, otherwise the evaluation is backed-up from successor nodes of n.

The distance is measured in the Number of Additional Picking Aisles that the order picker would have to visit in order to pick the product being evaluated. Note that the proposed distance measure is compatible with the accompanying-order selection heuristic for seed algorithms described in Subsection 3.2.2. For a single move in the search tree, the distance can either be one or zero. The tie breaking criterion is the true distance that the order picker would need to travel to pick all products currently on the picking list with addition of the product being evaluated, and return to the depot. The distance is calculated from the picking tour obtained using largest gap

routing strategy.

For heuristic evaluation of nodes at lookahead horizon we use a greedy algorithm. The static evaluation of the node is therefore the Number of Additional Picking Aisles of the remaining plan that is constructed using greedy approach, always choosing the product with smallest number of additional picking aisles and true travelling distance as a tie breaking criterion. This heuristic is pessimistic, as it never underestimates the number of additional picking aisles of the remaining plan. As shown in Sadikov and Bratko [17], pessimistic heuristics produce better results than optimistic when used with incomplete search methods such as RTA*.

When the planning stage is finished, the successor with the best backed-up evaluation is chosen and a move is made as a part of the execution stage. This move represents the algorithm choosing to pick up one particular product. The goal is found when the list of products and respective quantities is empty, i.e. when the customer order has been fulfilled. Usually, the RTA* algorithm stores already visited states. In warehouse domain such storage is not needed, as the states do not repeat themselves.

Increasing the search horizon (the lookahead parameter) increases the amount of computation per move, but decreases the number of moves required to solve the problem [18]. However, in our formulation of the problem, the number of moves depends solely on the number of products customer requested and therefore does not change with the lookahead parameter. It is therefore reasonable to expect that increasing the search horizon increases the amount of computation while the length of constructed picking tour decreases.

3.3.2 Hierarchical clustering

However, even with limited lookahead the search space grows exponentially with the size of the warehouse and the size of customer orders. We propose a solution to reduce branching factor in the search tree. We use clustering to group nearby products in clusters as part of the preprocessing and branch

only by one representative product from each cluster during plan construction which greatly reduces the algorithm search space. This can result in longer picking tours due to 'errors' that accumulate because of item clustering. We can control the size of the errors and consequently the length of the constructed picking tours by limiting the maximum size of the clusters.

We chose hierarchical clustering algorithm for grouping products into clusters. The reason for this is that hierarchical clustering is relatively simple algorithm with which we can easily limit the cluster size. The downside is high time complexity of $\mathcal{O}(n^3)$. However, this does not affect the picking tour construction times as it is a one-time operation.

Each product forms a separate cluster, $C_i = \{X_i\}, X_i \in \mathcal{U}$;
while *there is more than one cluster* **do**
  find closest clusters C_a and C_b, $d(C_a, C_b) = min_{u,v} d(C_u, C_v)$;
  join clusters C_a and C_b in cluster $C_{ab} = C_a \cup C_b$;
  replace clusters C_a and C_b with C_{ab};
end

Figure 3.6: Hierarchical clustering pseudocode

The basic algorithm is given in Figure 3.6. The distance between two products is defined as the distance that order picker has to travel to get from one product to the other. For calculating the distance between two clusters we used complete linkage method. The distance is computed as the maximum distance between a pair of products, one being in the first cluster, and the other being in the second (3.2).

$$d_{max}(C_u, C_v) = \max_{X,Y}\{d(X, Y) | X \in C_u, y \in C_v\} \tag{3.2}$$

Complete linkage enables us to effectively limit the maximum size of the cluster. This is achieved by interrupting the hierarchical clustering algorithm when the distance between the closest clusters surpasses the specified value.

Thus we get clusters which contain products that are at most the specified distance apart. We will refer to this value as the cut-off parameter.

Hierarchical clustering produces clusters of the same product type. It is applied to every product type in the warehouse, resulting in a number of clusters for every product type in the warehouse. A cluster of products is therefore a group of products of the same products type that are relatively close to each other.

When presented with a customer order which requests a product, instead of branching on every product of this type in the warehouse, we pick a random product from every cluster of this product type and branch on these. Instead of choosing a random product from a cluster, we could choose a product based on FIFO policy. By doing so we would partially address the concerns expressed by Van den Berg [16] (and discussed earlier in the thesis) regarding warehouse ageing.

3.4 Item reselection

Item reselection is an additional attempt at improving the constructed picking tours. Specific products to be picked are selected with RTA* algorithm based on individual customer order prior to the order batching. We attempt to improve the products selected based on the new information about the picking tour after the order batching was performed. The picking tour then includes a number of customer orders instead of only one.

The algorithm for item reselection is identical to the RTA* algorithm used to solve the item selection problem. The difference is the input to the algorithm. Instead of a single customer order, the algorithm receives a larger customer order consisting of all the customer orders in the batch. It is then applied to every batch individually. Clustering to reduce the branching factor can still be applied. The search space of the RTA* algorithm grows exponentially with the size of customer orders. As the batches are larger than individual customer orders the clustering of nearby products should results

in substantial reduction of computational time.

3.5 Storage policy

Storage policy is concerned with assigning products to storage locations when new products arrive in the warehouse. In the previous chapter we described three examples of storage policy: dedicated storage, randomized storage, and closest open location storage.

Dedicated storage policy assigns each arriving product to a fixed storage location (or a number of adjacent locations) that remains unchanged over a relatively long period of time.

Randomized storage assigns each arriving product to a random empty storage location in the warehouse.

Dedicated storage results in a *completely ordered warehouse*, i.e. every product type is stored in a single storage location or a number of adjacent storage locations. On the contrary, randomized storage results in a *completely unordered warehouse*, as the products of the same type are stored throughout the warehouse.

The randomized storage is often used in approaches studying problems other than the storage policy (for examples see [6], [11]). However, it is rarely modelled as a complete storage policy as that would require simulation of incoming products over a long period of time. Instead, it is simulated by generating a randomized initial warehouse inventory that is used for simulation experiments.

We use a similar approach of generating an initial warehouse inventory. However, in our experience, the completely randomized initial inventory is unrealistic compared to the warehouse inventories encountered in practice. Products often arrive in larger quantities. Even though the storage location assigned to them is chosen randomly, all the arriving products of the same type are stored at the randomly chosen storage location and adjacent storage locations if required. Therefore, the product of the the same type forms

clusters of individual products that are stored in different parts of the warehouse. If we extend previous definition of ordered warehouse to the idea of clusters, the warehouse is completely ordered if each product type is located only in one cluster in the warehouse. On the other hand, the warehouse is completely unordered when products of the same product type are stored in a number of clusters throughout the warehouse.

The pseudocode of the proposed algorithm for filling an empty warehouse is shown in Figure 3.7. It enables the control of warehouse order level using a parameter.

The algorithm expects the list of <*product_type, quantity*> pairs as its input. The pair <*product_type, quantity*> denotes the quantity of a product type to be stored in the warehouse. We discuss the random generation of the pairs in Chapter 4. Additionally, the algorithm requires the *sector_size_cutoff* parameter as its input. With the parameter we are able to control the order level of the warehouse.

The storage policy algorithm uses two important procedures. The *createSectors* procedure employs a hierarchical clustering algorithm to construct sectors, i.e. a group of storage locations that are relatively close to each other. Hierarchical clustering algorithm was already presented in Subsection 3.3.2 when we discussed our solution to the item selection problem. Here, the algorithm is employed in a similar fashion. However, instead of products it clusters storage locations. Additionally, we used average linkage method instead of complete linkage method for calculating the distance between two sectors. The distance is computed as the average distance between every pair of storage locations, one being in the first sector, and the other being in the second (3.3):

$$d(C_u, C_v) = \frac{\sum_{X \in C_u} \sum_{Y \in C_v} d(X, Y)}{|C_u||C_v|} \tag{3.3}$$

The hierarchical clustering algorithm is interrupted when the average distance between the closest sectors surpasses the *sector_size_cutoff* parameter.

```
Input: product_list: list of pairs <product_type, quantity>
Input: sector_size_cutoff : integer
sector_list = createSectors(sector_size_limit);
while product_list is not empty do
    sort product_list according to decreasing quantities;
    for <product_type, quantity> in product_list do
        sec = sector from sector_list with largest free capacity;
        if sec free capacity < quantity then
            assignStorageLocations(sec, <product_type, quantity>);
            <product_type, quantity> = <product_type, quantity - sec
            free capacity>
        end
        else if sec free capacity == quantity then
            assignStorageLocations(sec, <product_type, quantity>);
            remove <product_type, quantity> from product_list;
        end
        else
            sec = sector selected according to Best-Fit rule;
            assignStorageLocations(sec, <product_type, quantity>);
            remove <product_type, quantity> from product_list;
        end
    end
end
```

Figure 3.7: Storage algorithm pseudocode

With the parameter we are then able to control the size and the number of sectors that make up the warehouse and consequently the order level of the warehouse. Small sector size results in a high number of sectors existing in the warehouse. As the products are assigned to individual sectors, the size of the sector determines whether the products of the same type will be stored at adjacent storage locations or spread throughout the warehouse. When sectors are large enough to store all products of the same type, every product of that type is stored in adjacent storage locations.

The second procedure is *assignStorageLocations*. It stores the required number of products to the storage by choosing specific storage locations in the sector. Additionally, the procedure updates the free capacity of the sector.

Chapter 4

Experimental setup

We evaluated our approach to order picking optimization with experiments using a simulated warehouse environment. With the experiments we would like to show:

- the relation between increasing order in the warehouse and total length of picking tours,
- the feasibility of our approach with increasing size of the warehouse and number of customer orders,
- the effect of reselection on the length of picking tours,
- the effect of product clustering on the computation time.

The length of a picking tour is measured in meters. The order level of the warehouse is measured in meters as the cut-off parameter used when constructing sectors. The computation time is measured in seconds.

The experiments published in the order picking optimization literature are not conducted in a standard experimental warehouse setting and consequently vary greatly among researchers [3]. We attempt to emulate most of the warehouse setting from prominent researchers in the area. However, we were unable to avoid choosing some parameters to resemble warehouses encountered in practice to the best of our knowledge.

A summary of the warehouse parameters is presented in Table 4.1. The parameters are explained in greater detail in the following sections.

Parameter	Warehouse 1	Warehouse 2	Warehouse 3
Number of storage locations	960	1600	3200
Number of racks	240	400	800
Number of aisles	7	11	21
Number of product types	30	50	100
Fill percentage	75%	75%	75%
Order size	[2, 10]	[2, 10]	[2, 10]
Order picker capacity	24	24	24
Number of orders	30	50	100
Warehouse width	24 m	24 m	24 m
Warehouse length	26 m	42 m	82 m
Aisle width	2 m	2 m	2 m
Aisle length	20 m	20 m	20 m

Table 4.1: Parameters for the three warehouse situations.

Preliminary experiments showed that increasing the lookahead parameter did not significantly improve the length of picking tours, while increasing the computation time. Therefore, all of the experiments reported in this thesis were conducted with RTA* algorithm lookahead value of one.

Preliminary experiments also showed that clustering significantly decreased the computation time. However, as discussed in Section 3.3, cut-off values greater than zero resulted in longer picking tours. The total computation time of our optimization approach is short. Consequently, we did not use clustering above the cut-off parameter of zero so as to not increase the length of the constructed picking tours.

Clustering with a cut-off parameter of zero clusters together the products of the same type that are stored in the same rack or in the racks on the opposing sides of the aisle. This is because the order picker can pick them

from the same location, effectively making the distance between them zero. Therefore, clustering with a cut-off parameter of zero does not result in any increase in the picking tour lengths, while the effect on computation time is evaluated with the experiments.

4.1 Warehouse layout

In the low-level picker-to-parts warehouses used in the simulations, products are stored in a number of storage locations. Storage locations are of equal size (1m×1m×0.5m). Each storage location can store one product. The storage locations are located in a number of parallel aisles of the same width and length. Additionally, two cross aisles are used for switching the parallel aisles, one at the front and one at the back of the warehouse. The described warehouse layout is often referred to as *single-block layout* in order picking optimization literature and is a standard warehouse layout used for experiments [3, 4]. The depot, where order pickers start and end the picking tours, is located in the left corner of the front cross aisle. In the standard layout usually there is only one storage location per rack. In our warehouse layout we use racks with four storage locations stacked vertically. However, the pick time is equal for each of the stacked storage locations and therefore remains constant. An example of the described warehouse layout is shown in Figure 4.1.

The experiments are conducted in three warehouses of increasing size. The size is increased by adding a certain number of additional picking aisles to the warehouse, increasing the length of the warehouse while the width remains constant.

4.2 Warehouse inventory

The algorithm for filling an empty warehouse was described in Section 3.5. Here we briefly describe the generation of the <product_type, quantity>

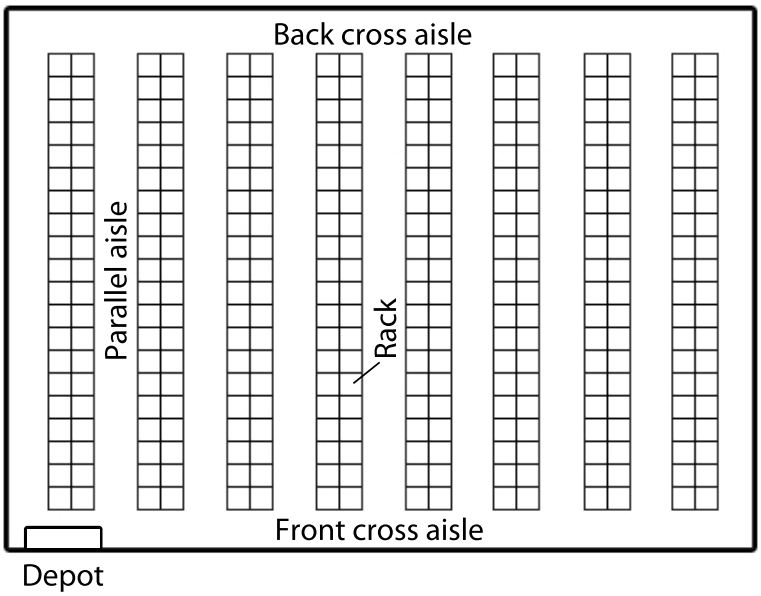

Figure 4.1: An example of warehouse layout.

pairs.

As the warehouses in practice are rarely completely full, the warehouse is only filled to 75% of its maximum capacity.

The number of pairs and consequently the number of product types in the warehouse is predetermined according to the warehouse size so that the mean quantity of the product type is the same in all warehouse sizes. The quantities for the product types are generated according to a normal distribution with the mean value of 24 and standard deviation of 10. If the quantity is less than or equal to zero no products of this product type are stored in the warehouse.

This results in larger warehouses storing proportionally larger number of product types, while the mean quantity of an individual product type does not change. The mean value of 24 was chosen to reflect the capacity of the picking device.

The set of pairs to be stored in the warehouse is the same for every experiment conducted in the warehouse of the same size. By using a random

seed with the storage policy algorithm presented in Section 3.5, we therefore construct an identical warehouse state at the beginning of every simulation conducted on the warehouse of the same size.

4.3 Customer orders

A customer order consists of a list of products and respective quantities to be picked. The size of the customer order, i.e. the sum of all quantities of products to be picked, is in the interval [2, 10] for all warehouse sizes. When generating customer orders the order size is determined according to a triangle distribution with the mean value of 6. The product types and quantities of a customer order are generated randomly with the limitation of the minimum and maximum order size and the actual availability of the products in the warehouse so that the customer order can be fulfilled.

A single simulation consists of picking a set of customer orders. The number of customer orders to be picked depends on the warehouse size. It was chosen so that on average the set of customer orders picked 25% of the warehouse inventory. While the number may seem large it is important to note that every storage location contains a single item that is picked as a whole, which may not hold true in practice.

Each data point in the experiment is evaluated on 50 sets of customer orders. The sets of customer orders are the same for warehouses of the equal size.

Chapter 5

Results

The results for each of the four experiments are presented in the following sections. All simulations were conducted on a PC with Intel i5-2430M 2.40 GHz processor and 8 GB of memory.

5.1 Warehouse order level

With the order level experiment we wanted to show the relation between the increasing order in the warehouse, as described in Section 3.5, and total length of picking tours.

The experiment was conducted in the medium-sized warehouse (Warehouse 2 in Table 4.1). We solved the order batching problem using three different approaches: FCFS, seed algorithm, and savings algorithm. No reselection was used, while product clustering had a cut-off parameter of 0.

The results are shown in Figure 5.1. It demonstrates the relation between average total picking tour length and increasing order level of the warehouse. It is evident that increasing the order level of the warehouse results in longer picking tours.

However, unusually, the picking tour lengths reach their peak at 2.5 meters and fall slightly when the order level is increased further. This is explained by the nature of the algorithm used for filling the warehouse with

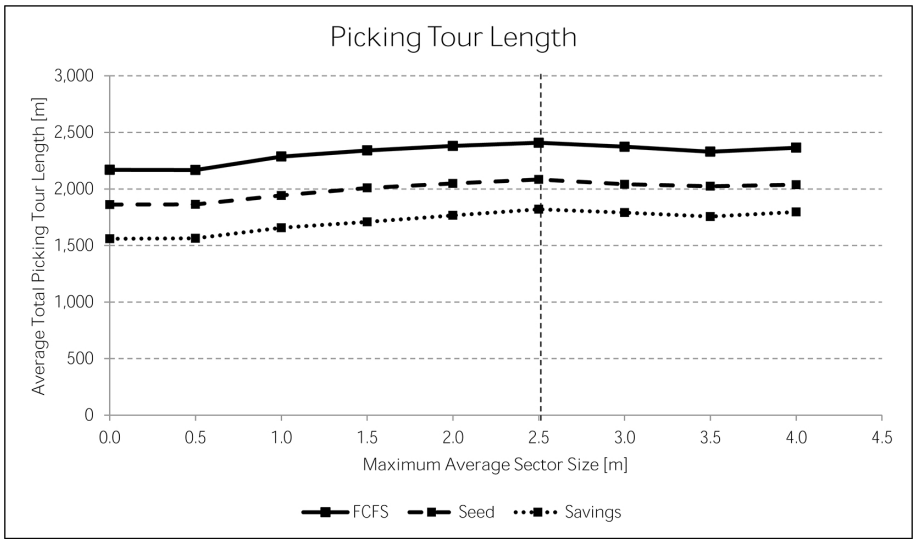

Figure 5.1: Average total picking tour lengths at increasing order level of the warehouse for 3 order batching algorithms. We sometimes substitute the term 'maximum average sector size' for simpler 'order level'.

products. When every product type is stored inside a single sector, we defined the warehouse as completely ordered.

This occurs exactly at order level 2.5 m, as it is evident from Figure 5.2. The figure presents an additional measure of average number of sectors per product type, normalized by the number of products. From order level 2.5 m and onwards, the average number of sectors reaches its minimum, meaning that every product is inside a single sector and warehouse is completely ordered. Therefore, the data points with order level greater than 2.5 meters are irrelevant to the experiment and are subject to anomalies introduced by the warehouse filling algorithm. For this reason, the remaining experiments presented in this chapter were conducted from the order level of 0 m to 2.5 m.

Additionally, in Figure 5.1 we can observe the behaviour of the three order batching algorithms. As expected, the savings algorithm performs best by consistently producing shorter picking tours. It is followed by the

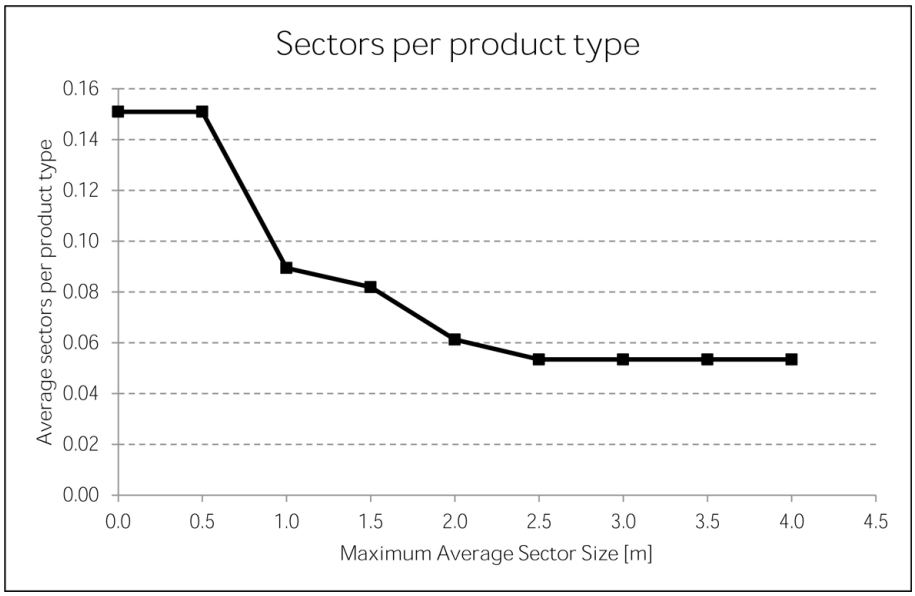

Figure 5.2: Average number of sectors per product type, normalized by the number of products, at increasing order level of the warehouse.

seed algorithm, while the FCFS, which was used as a baseline, produces the longest picking tours. The seed algorithm on average produces approximately 14% shorter picking tours compared to FCFS, while the savings algorithms produces approximately 26% shorter picking tours.

Figure 5.3 presents the reductions in average total picking tour length compared to the average picking tour length in completely ordered warehouse (at 2.5 m). As the values at 0 m and 0.5 m are almost identical we show only values at order levels from 0.5 m to 2 m, while 2.5 m is used as the reference point. From the figure we observe that reductions decrease with increasing order of the warehouse. The maximum average reduction of 14.4% is achieved using savings algorithm in completely unordered warehouse. This reduction is comparable with the reduction achieved by using seed algorithms instead of FCFS.

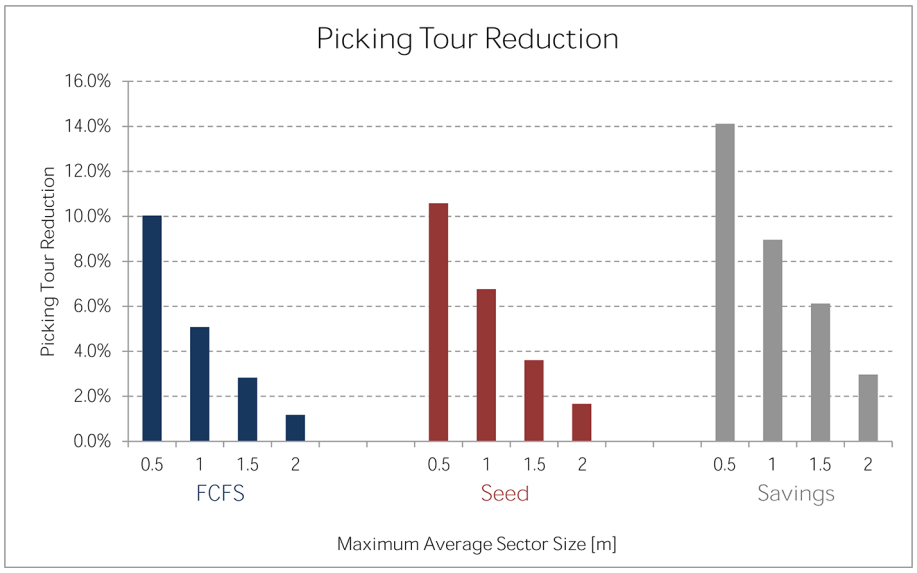

Figure 5.3: Picking tour length reductions at increasing order level compared to picking tour length in completely ordered warehouse for each of the three algorithms.

5.2 Feasibility

With the feasibility experiment we wanted to show the feasibility of our approach to the optimization of order picking with increasing size of the warehouse and number of customer orders.

The experiment was conducted in all three warehouse situations presented in Table 4.1. We solved the order batching problem using two competitive approaches: seed algorithm and savings algorithm. No reselection was used, while product clustering had a cut-off parameter of 0. The computation time was evaluated at different order levels, consisting of 450 simulations per warehouse - algorithm pair.

The results are shown in Figure 5.4. It shows that the computation time increased with warehouse size and number of orders. It is also clear that the computation time of the approach grows faster when savings algorithm is used for solving the order batching problem. Indeed, the largest average

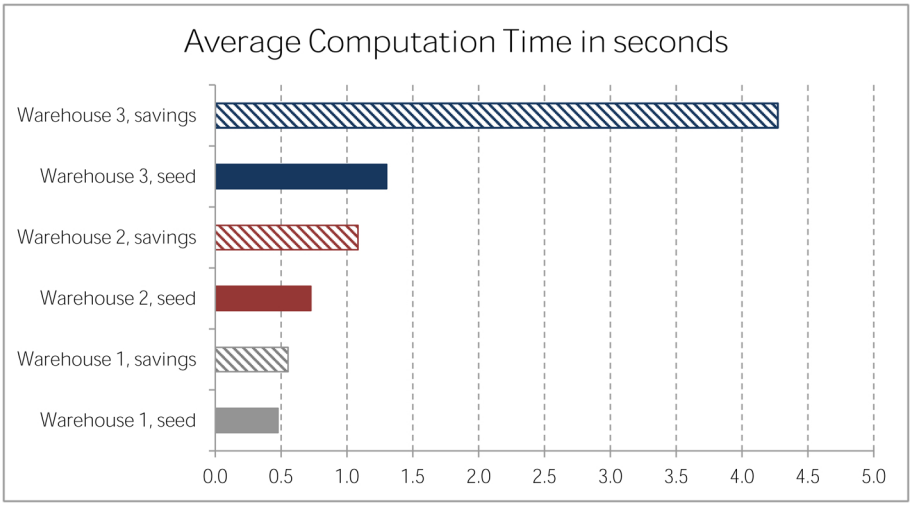

Figure 5.4: Average computation time at different warehouse sizes and order number for two competitive order batching algorithms.

computation time was measured when using the savings algorithm in the largest of the warehouses. However, it is important to note that even in this situation the average computation time is less than 5 seconds.

5.3 Item reselection

With the item reselection experiment we investigated the effect of reselection on the length of picking tours and computation time. The experiment was conducted in the medium-sized warehouse (Warehouse 2 in Table 4.1). We solved the order batching problem using two competitive approaches: seed algorithm and savings algorithm. Product clustering had a cut-off parameter of 0.

The results are shown in Figures 5.5 and 5.6. Figure 5.5 demonstrates the average total picking tour length with and without reselection for the two algorithms. Note that the chart does not start at zero and the differences may appear larger than they are. Despite this, it is clear that reselection does not improve picking tour lengths. On the contrary, we observe a significant

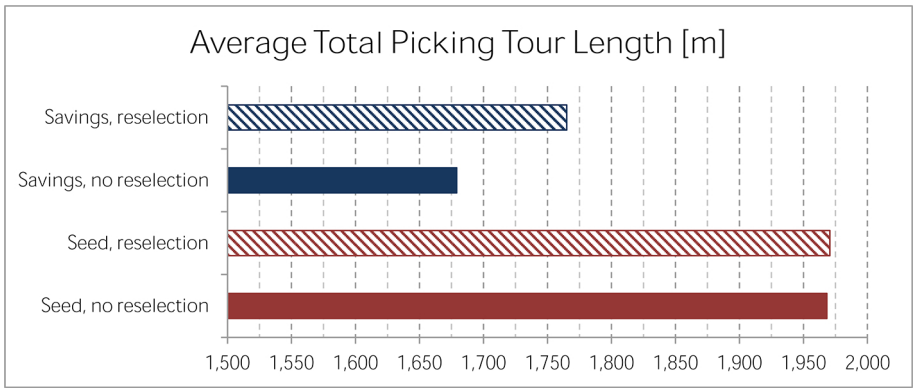

Figure 5.5: Average total picking tour length with and without reselection for two competitive order batching algorithms.

increase in picking tour lengths when reselection is used with the savings algorithm, while the difference is smaller with the seed algorithm.

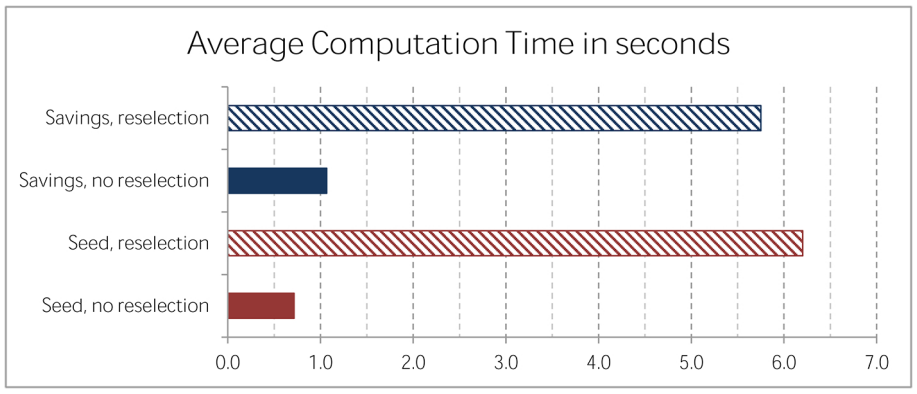

Figure 5.6: Average computation time with and without reselection for two competitive order batching algorithms.

Figure 5.6 shows the average computation time with and without reselection for the two algorithms. It is evident that reselection significantly increases the computation time with both algorithms. This is in large part due to increased search space of the RTA* algorithm when it is used for reselection of products for larger batches.

We can conclude that reselection does not provide any increase in quality of the picking tours while increasing the computation time significantly.

5.4 Product clustering

With the clustering experiment we examined the effect of product clustering on the computation time. We achieved this by conducting the experiment using clustering with cut-off parameter of zero, and using no clustering at all.

The experiment was conducted in the medium-sized warehouse (Warehouse 2 in Table 4.1). We solved the order batching problem using savings algorithm. No reselection was used.

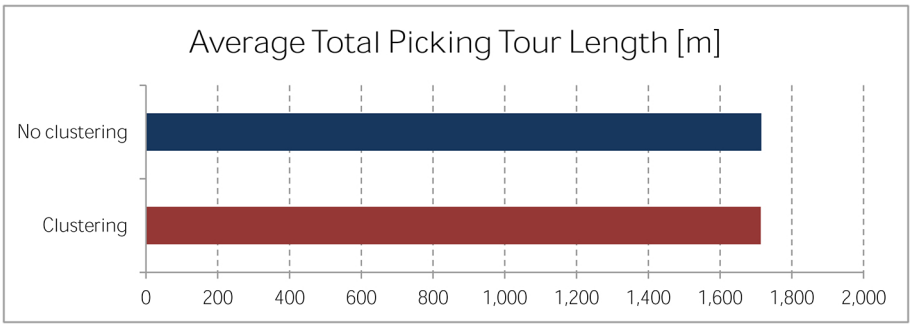

Figure 5.7: Average total picking tour length with and without clustering.

The results are shown in Figures 5.7 and 5.8. From Figure 5.7 we can observe that there is no significant difference between the length of picking tours generated with or without the use of clustering. In contrast, using clustering results in significantly less computation time to generate the picking tours, as it is evident from Figure 5.8. This can be explained by the decrease in the number of products to be evaluated by the RTA* during the item selection step. It results in an important search space reduction.

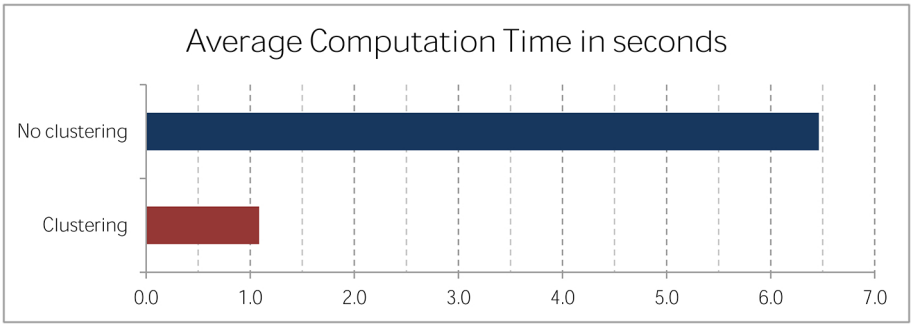

Figure 5.8: Average computation time with and without clustering.

Chapter 6

Discussion

As shown in Chapter 5, low order level of the warehouse results in up to 14% shorter picking tours compared to the high order level of the warehouse. Despite this, we noticed from our interaction with warehousing professionals that the warehouse managers are often hard to persuade to allow randomized storage of products. The reason for this seems to be that with conventional inventory tracking, the managers preferred to keep warehouses organized, which enabled the order pickers to easily find the required products. However, with the introduction of Warehouse Management Systems (WMS) and bar coding of products, the warehouse inventory is always up-to-date and can be effortlessly used to guide order pickers to correct product locations. Despite this, the warehouse managers seem reluctant to change old habits, which is the reason why randomized storage is not very popular.

Other storage policies that we did not investigate in this thesis may be superior to the randomized storage. An example of such policy is class-based storage. It assigns products to certain areas of the warehouse based on the demand frequency, i.e fast moving items are stored closer to the depot [2].

The feasibility experiment showed that our approach scales well with increasing size of the warehouse and number of customer orders. The size of the largest warehouse (Warehouse 3 presented in Table 4.1) is comparable with the largest experimental warehouses encountered in research literature.

Even though using savings algorithm for solving order batching problem in the largest warehouse resulted in a steep increase in computation time, the average computation time remained below five seconds. This makes the approach suitable for use in large warehouses in practice.

Our warehouse models assumed that only one product can be stored in a single storage location. This assumption often doesn't hold in practice. For example, a box of products or a pallet containing multiple products could be stored in a single storage location. It is often up to Warehouse Management System to represent the various granularity levels as products to be used in order picking optimization. However, even if each individual product is presented to our optimization approach, additional computation effort required for picking tour construction can be alleviated using product clustering. We observed that the quality of constructed picking tours will decrease in a linear fashion with increasing cut-off parameter. The computation time on the other hand decreased exponentially. Therefore, the quality of the picking tours can be traded for shorter computation time.

Unfortunately, we were unable to experimentally compare the proposed solution to the item selection problem to other approaches. The reason for this is that only one approach to solving it was proposed in relevant literature and it is incompatible with our complete approach to the order picking optimization. On the other hand, comparing it with existing de facto solution, the FIFO policy, would be unfair. Our approach would produce far superior results in terms of picking tour length, while failing to account for the other important part of the FIFO policy - warehouse ageing. The proposed algorithm, however, enables the use of a combined heuristic that would account for the time product is stored in the warehouse. Determining the success of such a heuristic would require simulating the warehouse operations over a long period of time to measure the possible inventory ageing. We did not attempt to simulate such advanced warehouse environments in this thesis and leave this as future work.

Product reselection was shown to be ineffective. It resulted in construc-

tion of picking tours of equal or greater length than without reselection while significantly increasing the computation time. Additional approaches to improving item reselection is an interesting topic for further work.

Chapter 7

Conclusions

The conventional approach to order picking optimization divides the process in three main decision areas: storage policy, order consolidation policy, and routing policy.

In this thesis, we extended this approach by proposing a solution to the item selection problem using RTA* heuristic search algorithm. The item selection problem is concerned with selecting a single product to pick from multiple products throughout the warehouse so as to satisfy a customer order. However, the search space of the RTA* algorithm grows exponentially with the warehouse and order size. We introduced product clustering using hierarchical clustering algorithm as a method of decreasing RTA* search space. Alongside established solutions to the order batching and picker routing problems we integrated this into a complete sequential approach to order picking optimization.

We evaluated our approach using experiments in simulated warehouse environment. We implemented a randomized storage algorithm that produces a more realistic initial warehouse inventory for the simulation experiments. It enables the control of the order level of the warehouse. We demonstrated that low order level of the warehouse is to be preferred as it results in up to 14% shorter picking tours compared to high warehouse order level.

We showed that our approach scales well with the increasing warehouse

size and number of orders, on average constructing picking tours in less than five seconds even in large warehouses. Product clustering was shown to be an important method for reducing the computation time in all warehouse sizes.

However, we were unable to compare the proposed item selection algorithm with de facto FIFO policy as that would require simulating the warehouse environment over a long period of time. This is an interesting topic for future work.

Bibliography

[1] E. Frazelle, *World-Class Warehousing and Material Handling.* New York: McGraw-Hill, 2001.

[2] R. de Koster, T. Le-Duc, and K. J. Roodbergen, "Design and control of warehouse order picking: A literature review," *European Journal of Operational Research*, vol. 182, no. 2, pp. 481 – 501, 2007.

[3] S. Henn, S. Koch, and G. Wäscher, "Order batching in order picking warehouses: A survey of solution approaches," in *Warehousing in the Global Supply Chain - Advanced Models, Tools and Applications for Storage Systems.* London: Springer-Verlag, 2012, pp. 105 – 137.

[4] G. Wäscher, "Order picking: a survey of planning problems and methods," in *Supply Chain Management and Reverse Logistics.* Berlin: Springer, 2004, pp. 324 – 370.

[5] D. M. Lambert, J. R. Stock, and L. M. Ellram, *Fundamentals of Logistics Management.* Boston, MA: Irwin/McGraw-Hill, 1998.

[6] R. de Koster, E. Van der Poort, and M. Wolters, "Efficient orderbatching methods in warehouses," *International Journal of Production Research*, vol. 37, no. 7, pp. 1479–1504, 1999.

[7] F. Dallari, G. Marchet, and M. Melacini, "Design of order picking system," *The International Journal of Advanced Manufacturing Technology*, vol. 42, pp. 1–12, 2009.

[8] R. de Koster, K. J. Roodbergen, and R. Van Voorden, "Reduction of walking time in the distribution center of De Bijenkorf," in *New trends in distribution logistics*. Berlin: Springer, 1999, pp. 215–234.

[9] J. A. Tompkins, J. A. White, Y. A. Bozer, and J. Tanchoco, *Facilities Planning*. New Jersey: Wiley, 2003.

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

[11] Y.-C. Ho and Y.-Y. Tseng, "A study on order-batching methods of order-picking in a distribution centre with two cross-aisles," *International Journal of Production Research*, vol. 44, no. 17, pp. 3391–3417, 2006.

[12] G. Clarke and J. W. Wright, "Scheduling of vehicles from a central depot to a number of delivery points," *Operations Research*, vol. 12, pp. 568–581, 1964.

[13] H. D. Ratliff and A. S. Rosenthal, "Order-picking in a rectangular warehouse: A solvable case of the traveling salesman problem," *Operations Research*, vol. 31, pp. 507–521, 1983.

[14] C. G. Petersen, "An evaluation of order picking routeing policies," *International Journal of Operations and Production Management*, vol. 17, pp. 1098 – 1111, 1997.

[15] R. L. Daniels, J. L. Rummel, and R. Schantz, "A model for warehouse order picking," *European Journal of Operational Research*, vol. 105, no. 1, pp. 1 – 17, 1998.

[16] J. P. Van den Berg, "A literature survey on planning and control of warehousing systems," *IIE Transactions*, vol. 31, pp. 751–762, 1999.

[17] A. Sadikov and I. Bratko, "Pessimistic heuristics beat optimistic ones in real-time search," in *Proceedings of the 2006 conference on ECAI 2006: 17th European Conference on Artificial Intelligence August 29 – September 1, 2006, Riva del Garda, Italy.* Amsterdam, The Netherlands: IOS Press, 2006, pp. 148–152.

[18] R. E. Korf, "Real-time heuristic search," *Artificial Intelligence*, vol. 42, no. 2-3, pp. 189 – 211, 1990.