**KAUNAS UNIVERSITY OF TECHNOLOGY**

**FACULTY OF MECHANICAL ENGINEERING AND DESIGN**

**Arnoldas Armonas**

# DEVELOPMENT OF WAREHOUSE ORDERS BATCHING IMPROVEMENT ALGORITHM

Master’s Degree Final Project

**Supervisor**
Assoc. prof. dr. Kazimieras Juzėnas

**KAUNAS, 2017**

**KAUNAS UNIVERSITY OF TECHNOLOGY**

**FACULTY OF MECHANICAL ENGINEERING AND DESIGN**

# DEVELOPMENT OF WAREHOUSE ORDERS BATCHING IMPROVEMENT ALGORITHM

Master's Degree Final Project

**Industrial Engineering and Management (621H77003)**

**Supervisor**

Assoc. prof. dr. Kazimieras Juzėnas

**Reviewer**

Assoc. prof. dr. Giedrius Janušas

**Project made by**

Arnoldas Armonas

**KAUNAS, 2017**

**KAUNAS UN IVERSITY OF TECHNOLOGY**

**FACULTY OF MECHANICAL ENGINEERING AND DESIGN**

**Approved:**

| | |
|---|---|
| Head of Production engineering Department | ______________ *(Signature, date)* |
| | ***Kazimieras Juzėnas*** *(Name, Surname)* |

**MASTER STUDIES FINAL PROJECT TASK ASSIGNMENT**
**Study programme INDUSTRIAL ENGINEERING AND MANAGEMENT**

The final project of Master studies to gain the master qualification degree, is research or applied type project, for completion and defence of which 30 credits are assigned. The final project of the student must demonstrate the deepened and enlarged knowledge acquired in the main studies, also gained skills to formulate and solve an actual problem having limited and (or) contradictory information, independently conduct scientific or applied analysis and properly interpret data. By completing and defending the final project Master studies student must demonstrate the creativity, ability to apply fundamental knowledge, understanding of social and commercial environment, Legal Acts and financial possibilities, show the information search skills, ability to carry out the qualified analysis, use numerical methods, applied software, common information technologies and correct language, ability to formulate proper conclusions.

1. Title of the Project

| |
|---|
| Development of Warehouse Orders Batching Improvement Algorithm |

Approved by the Dean Order No. V25-11-8, 21 April 2017

2. Aim of the project

| |
|---|
| To develop the heuristic algorithm for order batching problem that could be used in warehouses. |

3. Structure of the project

| |
|---|
| Analysis of existing algorithms;<br>Development of a new algorithm;<br>Testing new algorithm against already existing algorithms in simplified warehouse simulation;<br>Testing new algorithm in real warehouse simulation;<br>Results and Conclusions. |

4. Requirements and conditions

| |
|---|
| None |

5. This task assignment is an integral part of the final project

6. Project submission deadline: 2017 – 05 – 24

| | | |
|---|---|---|
| Student | ______ Arnoldas Armonas_____ (*Name, Surname of the Student*) | ____________ (*Signature, date*) |
| Supervisor | Assoc. prof. dr. Kazimieras Juzėnas (*Position, Name, Surname*) | ____________ (*Signature, date*) |

**KAUNAS UNIVERSITY OF TECHNOLOGY**

Faculty of Mechanical Engineering and Design
(Faculty)

Arnoldas Armonas
(Student's name, surname)

Industrial Engineering and Management 621H77003
(Title and code of study programme)

"Development of Warehouse Orders Batching Improvement Algorithm"

# DECLARATION OF ACADEMIC INTEGRITY

14 June 2017
Kaunas

I confirm that the final project of mine, **Arnoldas Armonas**, on the subject "Development of Warehouse Orders Batching Improvement Algorithm" is written completely by myself; all the provided data and research results are correct and have been obtained honestly. None of the parts of this thesis have been plagiarized from any printed, Internet-based or otherwise recorded sources. All direct and indirect quotations from external resources are indicated in the list of references. No monetary funds (unless required by law) have been paid to anyone for any contribution to this thesis.

I fully and completely understand that any discovery of any manifestations/case/facts of dishonesty inevitably results in me incurring a penalty according to the procedure(s) effective at Kaunas University of Technology.

*(name and surname filled in by hand)* *(signature)*

Armonas, Arnoldas. Development of Warehouse Orders Batching Improvement Algorithm: Master’s thesis. Supervisor assoc. prof. Kazimieras Juzėnas. The Faculty of Mechanical Engineering and Design, Kaunas University of Technology.

Research area and field: Technological Sciences, Production Engineering

Key words: Order, Warehouse, Batching, Algorithm, Heuristic, Picking, Development, Genetic

Kaunas, 2017. 68 p.

## Summary

Orders picking is one of the key functions of the warehouse. Analysis shows that the most of the picking is done manually. Therefore, orders picking is very labor-intensive process.

The aim of this work is to develop an algorithm, which would improve orders picking process in warehouses. By creating batches, this algorithm should shorten the travel distance needed to pick all the items of the orders.

Thesis consist of three parts. In the first part, literature analysis is done. The most often used and best performing algorithms for order batching are identified and compared. Also, two benchmark algorithms are selected.

In the second part a new algorithm is developed. The new algorithm is population based Genetic algorithm. The performance of the developed algorithm and benchmark algorithms is than measured and compared in the simulated simplified warehouse environment. Results show that new algorithm generates better order batches than benchmark algorithms. However, the Genetic algorithm needs more time to complete the computations. Based on the insights on the results the Genetic algorithm is updated.

The real warehouse simulation environment is than configured in the third part. The orders batching algorithm used in this warehouse is identified and simulated. The performance of the Genetic algorithm developed in this work is compared against the algorithm which is used in this warehouse. Results of the experiment show that Genetic algorithm would provide 17% shorter travel distances on average, when compared to current solution used in the warehouse.

Arnoldas Armonas. Sandėlio užsakymų surinkimo proceso tobulinimo algoritmo kūrimas: Magistro baigiamasis darbas. Darbo vadovas doc. dr. Kazimieras Juzėnas. Mechanikos inžinerijos ir dizaino fakultetas, Kauno Technologijos Universitetas.

Tyrimų sritis: technologijų mokslai, gamybos inžinerija.

Esminiai žodžiai: užsakymas, sandėlis, rūšiavimas, algoritmas, euristika, surinkimas, kūrimas, genetinis.

Kaunas, 2017. 68 p.

## Santrauka

Užsakymų surinkimas yra viena pagrindinių sandėlio funkcijų. Analizė rodo, kad dažniausiai, surinkimo procesas atliekamas rankiniu būdu. Taigi, užsakymų surinkimas yra labai daug darbo reikalaujantis procesas.

Šio darbo tikslas yra sukurti algoritmą, kuris pagerintų užsakymų surinkimo procesą sandėliuose. Grupuodamas užsakymus, šis algoritmas, turėtų sutrumpinti kelią, reikalingą surinkti visoms prekėms užsakymuose.

Darbas susideda iš trijų dalių. Pirmojoje atliekama literatūros analizė. Išskiriami ir palyginami dažniausiai naudojami ir geriausiai veikiantys užsakymų grupavimo algoritmai. Taip pat, išrenkami du etaloniniai algoritmai.

Antrojoje dalyje kuriamas naujas algoritmas. Naujas algoritmas yra populiacija paremtas Genetinis algoritmas. Po to, šio ir etaloninių algoritmų veikimas išmatuojamas ir palyginamas dirbtinėje supaprastinto sandėlio aplinkoje. Rezultatai parodo, kad naujasis algoritmas sukuria geresnes užsakymų grupes nei etaloniniai algoritmai. Visgi, Genetiniam algoritmui reikia daugiau laiko užbaigti skaičiavimus. Remiantis šios dalies rezultatais, Genetinis algoritmas koreguojamas.

Trečiojoje dalyje sukonfigūruota tikru sandėliu paremta dirbtinė aplinka. Taip pat, nustatytas ir įgyvendintas šiame sandėlyje naudojamas užsakymų rūšiavimo algoritmas. Šiame darbe sukurto Genetinio algoritmo rezultatai palyginti su sandėlyje naudojamo algoritmo rezultatais. Eksperimento metu paaiškėjo, kad Genetinis algoritmas vidutiniškai generuoja 17% trumpesnius maršrutus nei šuo metu sandėlyje naudojamas sprendimas.

# Table of Contents

# List of Tables

# List of Figures

# Introduction

The present trend in manufacturing companies is to reduce the delivery time to the lowest possible, combined with a high array of options available (mass customization), and smaller and more frequent orders [1]. As a result of this trend, the necessity for fast and reliable warehouses and distribution centers is essential for the organizations seeking to succeed among competitors.

Main functions of warehouses are receiving, storage, order picking and shipping. Among these functions, order picking is known to be the most labor and cost intensive. Order picking can be defined as the process of warehouse by which a small number of goods are extracted from a warehousing system, to satisfy a number of independent customer orders [2]. The cost of order picking is estimated to reach from 55% to 70% of the total warehouse operating expense [3] [4]. As a warehouse function, order picking is critical to each supply chain, since underperformance results in an unsatisfactory customer service (long processing and delivery times, incorrect shipments) and high costs (labor cost, cost of additional and/or emergency shipments).

Two basic types of order picking systems can be distinguished: manual order picking systems and technical systems. As their names suggest, the first one uses human operators and the second one uses machinery solely, therefore is completely automated. Manual order picking systems can be further differentiated into picker-to-parts systems and parts-to-pickers systems [5].

In picker-to-parts systems the order picker walks or rides through warehouse area, stops at storage locations of the items needed to pick and takes required units of those items. In low level picker-to-parts systems the items usually stored on pallets or bins placed on the warehouse floor, or on low level racks which are directly accessible by order picker from warehouse floor. In high level picker-to-parts systems the picking area consists of high storage racks. In this case picker can access items using crane or other vehicle that can move and elevate order picker to pick needed items.

In parts-to-picker systems automated storage and retrieval system loads items from warehouse and delivers them to the transfer site. Here stationary order picker takes requested items and materials handling system returns to its location in warehouse.

The most of warehouses use picker-to-parts systems [6] [7]. According to Koster, more than 80% of all order picking systems in Western Europe was low level picker-to-parts systems in 2007 [5]. The study of Warehouse Excellence (2011) also supports this. Study showed that 183 of 220 surveyed companies (83%) use picker-to-parts systems [6]. According to survey made by Napolitano in 2012, 80% of all surveyed companies use manual picking systems [8]. Despite many efforts towards more automated warehouses the order picking process as a whole remains a very labor-intensive operation, due to simplicity and flexibility of manual picking systems [9].

It is clear to see that order picking operation has the highest potential for improvements among all the basic warehouse activities. There are different order picking techniques to do so, however this work is focused solely on batch picking.

Batch picking is a technique when one picker picks a group (batch) of orders at the same time, one item at a time. This is advantageous when there are multiple orders with the same items or items stored near each other.  When that occurs, the order picker only needs to travel to the pick location for that specific item once, in order to fill the multiple orders [10].

Items are collected on tours through the warehouse, where number of stops on each tour is limited by the available capacity of the picking device and by capacity requirements of the items to be picked. Customer orders are combined into picking orders until capacity of picking device is exhausted. For the definition of the capacity various criteria are used. The capacity is expressed in number of items [11]. Splitting of customer orders is usually prohibited because it would lead to additional sorting operation. Having this in mind, the order batching problem could be defined as follows: how, given finite capacity of picking device, can a given set of customer orders with known storage locations be grouped into picking orders such that total length of all picker tours would be minimized [12].

Orders batching problem is known to be the NP hard type problem [13]. As explained by Jeff Erickson, solving NP hard problem is the same as finding mathematical proof, that dogs can’t speak fluent English. The fact that no one has ever heard a dog speak English is evidence, as are the hundreds of examinations of dogs that lacked the proper mouth shape and brainpower, but mere evidence is not a mathematical proof [14]. There is no known way to solve NP hard problem other than checking all possible solutions.

Direct solution approaches for the orders batching problem have limited use due to nature of problem. In real-life situations, this problem is usually solved using heuristics. A heuristic is a shortcut that allows to solve problems and make judgments quickly and efficiently. However, heuristics, most often does not find optimal but, rather good enough solutions for the problems.

The aim of this work is to develop the heuristic algorithm for order batching problem that could be used in warehouses. This algorithm should generate solutions that would be as good or better than heuristics currently used in warehouses and also could be computed in reasonable amount of time. To reach this goal, further tasks should be completed:

1. Most often used and best performing solutions should be chosen for validation.
2. New algorithm for orders batching should be implemented.
3. The performance of new and benchmark algorithms should be compared in simulated testing environment.
4. Testing environment should be configured based on real warehouse example. New and currently in this warehouse used solutions should be tested and compared.

# 1. Overview of Orders Batching Algorithms

As mentioned before, direct order batching problem solving approaches are not used in real warehouse situation very often. Heuristic approaches are used instead. These approaches could be classified into two groups:

1. Constructive heuristics;
2. Metaheuristics.

## 1.1 Constructive Heuristics

A constructive heuristic is a type of heuristic methods which starts with an empty solution and repeatedly extends the current solution until a complete solution is obtained. [15].

### 1.1.1 Priority Based Heuristics

The most often in real warehouse situation used solutions are simple constructive heuristics known as priority based heuristics. These are simple solutions usually consisting of two steps: orders prioritization and orders batching.

There are a lot of rules that could be used for orders prioritization, but probably the most straightforward rule is the First Come First Served (FCFS) rule. As the name suggests priorities are set to orders as they come to warehouse. The earlier order come the earlier it will be picked. Orders are added to the batch based on their arrival time. One batch is closed and another is opened when new orders does not fit into picking device. This heuristic generates random results, however as noted in literature, it is often used as benchmark for experiments.

FCFS heuristic uses next-fit orders assignment to batches rule. It creates one batch at a time, and opens new one, when new order can’t be fitted into already opened batch. Orders could be also assigned to batches simultaneously instead of sequentially using first-fit and best-fit rules. According to first-fit rule, batches are indexed in the order they were opened. New order is assigned to a batch which has smallest number of items and still have sufficient capacity to add items from new order [16]. The best-fit rule sets order to a batch which has least remaining capacity and still can accept items from new order [12].

More complex priority based batching rule was suggested by Pan and Liu [17]. Based on works of Gibson and Sharp, they suggested the application of six-dimensional space filling curve for prioritization. Dimensions used for this prioritization represents parameters of items storage location. Each item in customer order is mapped onto the curve where its position could be then described as θ value (could be a value from 0 to 1). Order having largest θ is set with highest priority. To optimize this solution, Ruben and Jacobs suggested that customer orders should be sorted according to the order envelope. Order envelope is defined by two numbers, where first one represents the leftmost aisle of the order and second represents rightmost aisle of the order [16]. The general principle of priority based heuristics is shown in Figure 1.1.

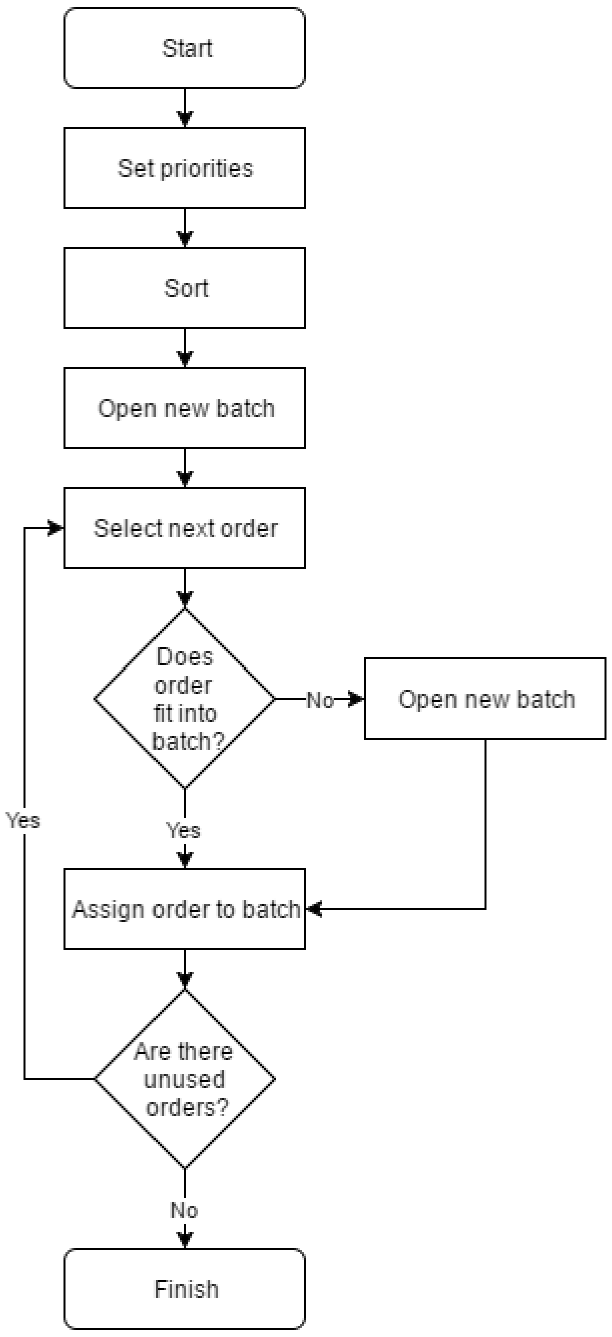


**Figure 1.1. General principle of priority based heuristics**

### 1.1.2 Seed Heuristics

Seed heuristics was first mentioned in 1981 by Elsayed [18]. This algorithm sequentially generates batches in two steps. First step is seed selection. During this step, an initial order which is not assigned to any other batch is chosen as seed for a new batch. Seed could also be not one, but combination of orders. There are many seed selection rules that are used and could be applied from simple rules like random seeding to advanced rules like smallest aisle-exponential weight sum [19].

After seed is selected, other orders are added to the batch according to order congruency rule, which measures distance from the seed to new customer order. Base idea behind seed algorithm is shown in Figure 1.2.

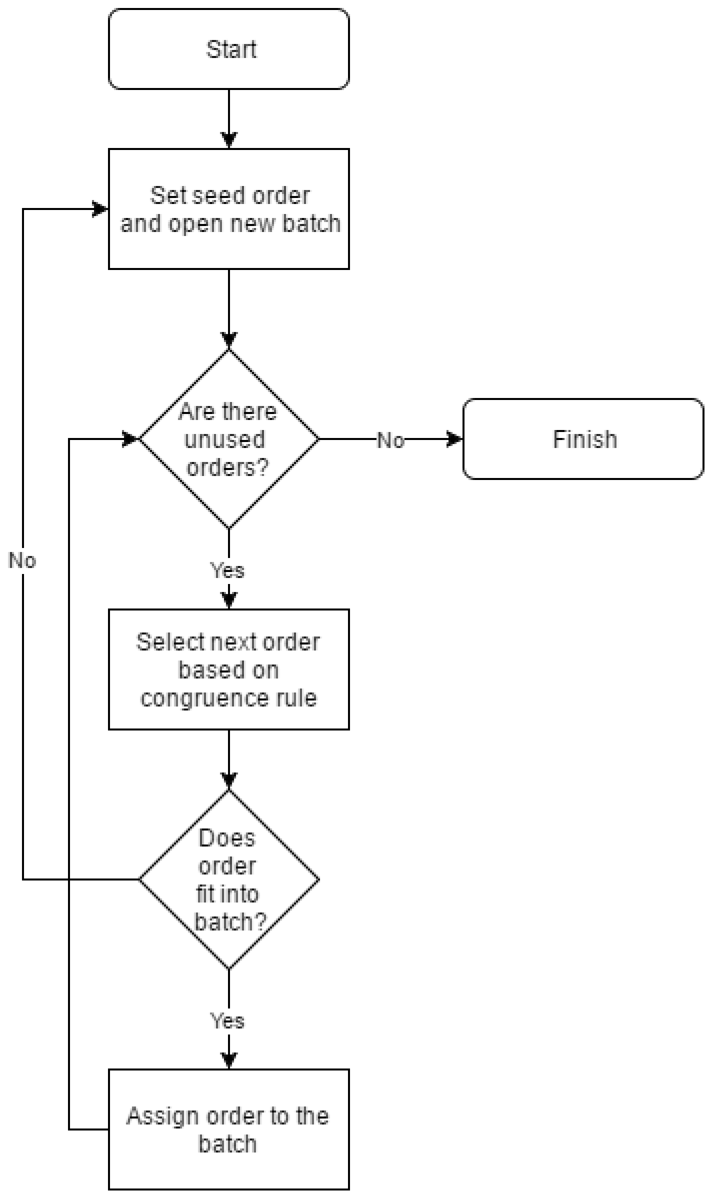

Start
Set seed order and open new batch
Are there unused orders?
No
Finish
Yes
Select next order based on congruence rule
Does order fit into batch?
No
Yes
Assign order to the batch

**Figure 1.2. General principle of seed heuristic**

### 1.1.3 Savings Algorithms

All savings heuristics are based on Clarke and Wright algorithm developed to solve vehicle routing problem. This algorithm was adapted to order batching problem in several ways. In first adaptation of this algorithm (called CW1), savings of all two orders combinations are calculated. Savings is a distance that is saved, when those orders are collected at the same tour instead of collecting them separately. Starting with the pair of orders that have highest savings, orders are assigned to batches.

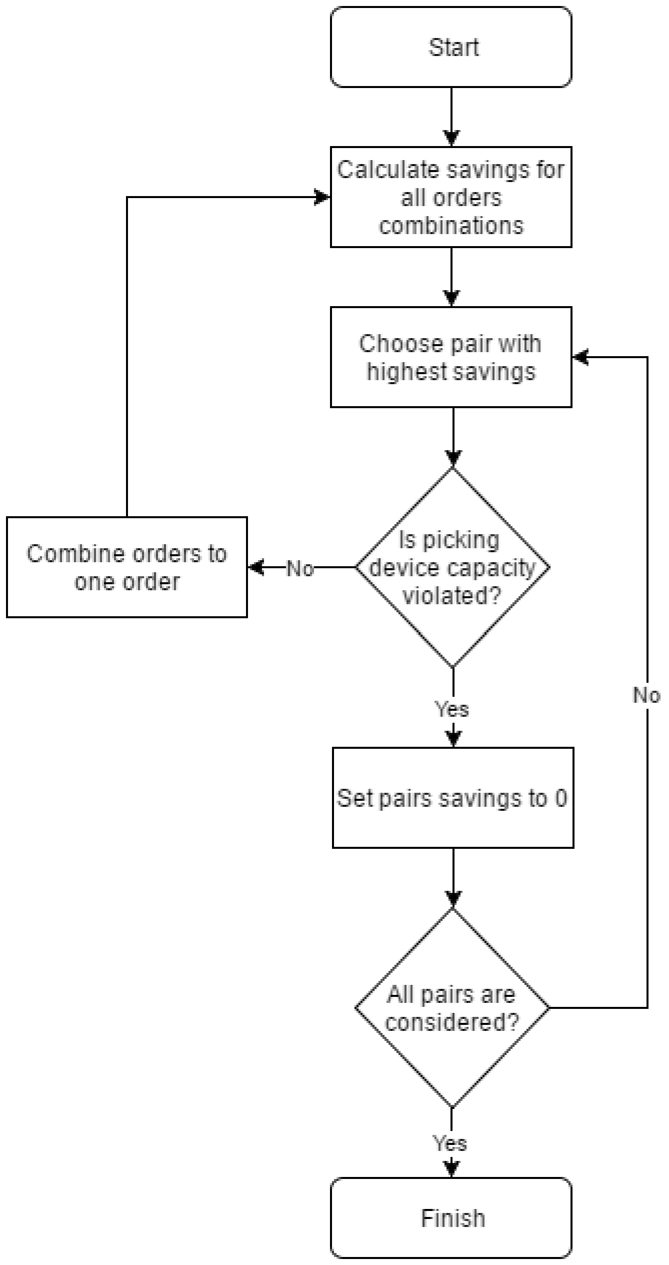

Start
Calculate savings for all orders combinations
Choose pair with highest savings
Is picking device capacity violated?
No
Combine orders to one order
Yes
Set pairs savings to 0
All pairs are considered?
No
Yes
Finish

**Figure 1.3. CW2 savings algorithm**

Assignment of orders could end up in three situations:

1. None of orders in pair has been assigned to batch. In this case, new batch will be started, and both orders will be assigned to that batch.
2. One of orders is already assigned to the batch. If batch capacity is sufficient, second orders is also assigned to this batch. If not, order is skipped and other pair of orders is considered.
3. Both orders have already been assigned to batches. In this case, next pair of orders is considered.

In the end, all unassigned orders will be added to individual batches [20].

The main problem with this algorithm is that it will generate a large number of possible batches, which makes computing time unacceptable for real life situations. To prevent this problem savings values of orders that are not assigned to batches are reduced by some constant value. After such modifications, heuristic will be tending to select order pairs, in which at least one order is already assigned to batch [20].

In the second adaptation of same heuristic (called CW2), savings are recalculated each time new assignment of order to batch has been made. Orders that are in batches are excluded from calculations.

Another saving algorithm based on the same idea is called Equals [20]. This heuristic uses pair of orders with highest saving as the initial seed for the batch. Then if picking device's capacity is not violated, other order that will make savings of the batch highest assigned to the batch. This is repeated until capacity of picking device is violated. After that new batch is opened and process is repeated.

Savings heuristic algorithm is shown in Figure 1.3.

## 1.2 Metaheuristics

Metaheuristics are stochastic optimization algorithms and techniques, which uses some degree of randomness to find solutions for problems [21].

### 1.2.1 Local Search

The main idea behind Local Search algorithms is to explore the neighbor solutions to identify new solution with smaller objective function value. If there is a solution S, neighbor solution could be found by applying a single local transformation ("move") for S. Local Search algorithms usually generates a sequence of solutions, which could be represented as S0, S1, S2…, where each solution has smaller objective function value then previous solution. The classical Local Search algorithm finishes when no smaller objective function value could be found. However, this leads to a problem, when only local minimal solution is found, and often this solution is far from optimal [21].

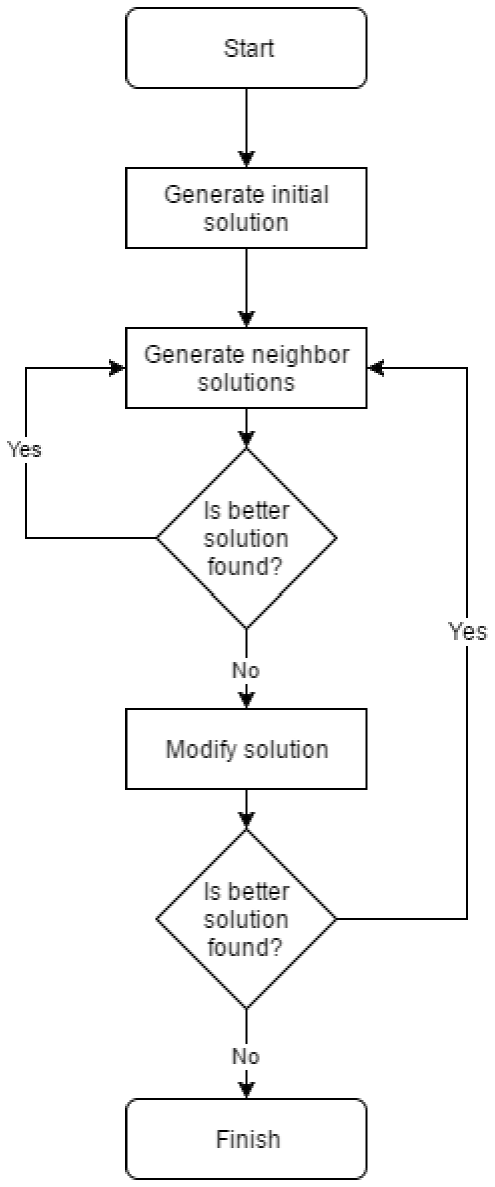

Start
Generate initial solution
Generate neighbor solutions
Yes
Is better solution found?
No
Modify solution
Yes
Is better solution found?
No
Finish

**Figure 1.4. Local search with solution modification stage**

The first step in their approach is to generate initial solution. Usually simple first come first served (FCFS) rule for that. Then neighbor solutions where found by so-called swap moves. In swap move orders from different batches where interchanged if this move is valid in terms of picking device capacity. Then, if there is a solution, which has better objective function value, it is accepted as current best solution. When no better solutions could be found, the best solution is modified by three operations. In each operation three customer orders from three different batches are randomly interchanged. Solution created after these three operations is then taken to local search phase again. This algorithm loops for a predetermined number of iterations. Base idea of this algorithm is represented in Figure 1.4.

Later Henn suggested, that order batching problem could be solved by iterated local search (ILS) [11]. This algorithm intensifies search for better solution around local minimal. Similar to previous approach, this algorithm consists of two main stages local search and modification. The main difference from previous approach is that for a new solution created by local search to be passed to next stage, it has to pass acceptance function. If it does not, previous solution remains the best solution and modifications are applied to this solution again. The algorithm is repeated for determined count of iterations.

The first step in this approach is to generate initial solution. As suggested by authors it is done using FCFS rule. When solution is generated, the local search operation is started. The local search here consists of two moves swap and shift. As defined previously, swap move is interchange of two orders in two batches if it is allowed by picking device capacity. Shift move is placing one random order to another batch if it is allowed by picking device constraints. Local search starts with swap moves and continues until no better solutions could be found. Then shift moves are applied until no further improvements could be made. Then again swap moves are applied. Swap and shift moves alternates until no better solution could be found. In the modification stage two batches are selected randomly and random number of order are moved from one to the other and vice versa. If orders do not fit into batch due to picking device capacity violation, they are moved to the new batch. A new solution is accepted if its objective function value is better than the one from current best solution.

Another approach to orders batching problem using Variable Local Search (VLS) is made by Albareda-Sambola and others. They defined three different neighborhoods created by:

1. Assignment of one order to another batch (shift move);
2. Assignment of up to two orders from one batch to other batches;
3. Assignment of up to two orders from one or two batches to one or two batches.

First algorithm explores the solution neighborhoods using first move. When no better solution could be found, algorithm uses second move. When second move could not generate better solution, third move is used. When third move is finished and current best solution is identified, algorithm repeats all process using this new best solution. When no better solution could be found, algorithm stops [22].

### 1.2.2 Tabu Search

Tabu search algorithm is developed by Glover in 1986. The main idea of this algorithm is very similar to local search algorithm, it searches for better solution in solution neighborhood. In addition to that, tabu search algorithm aims to simulate human memory using so called tabu list. This list stores previous solutions.  Tabu search algorithm is represented in Figure 1.5

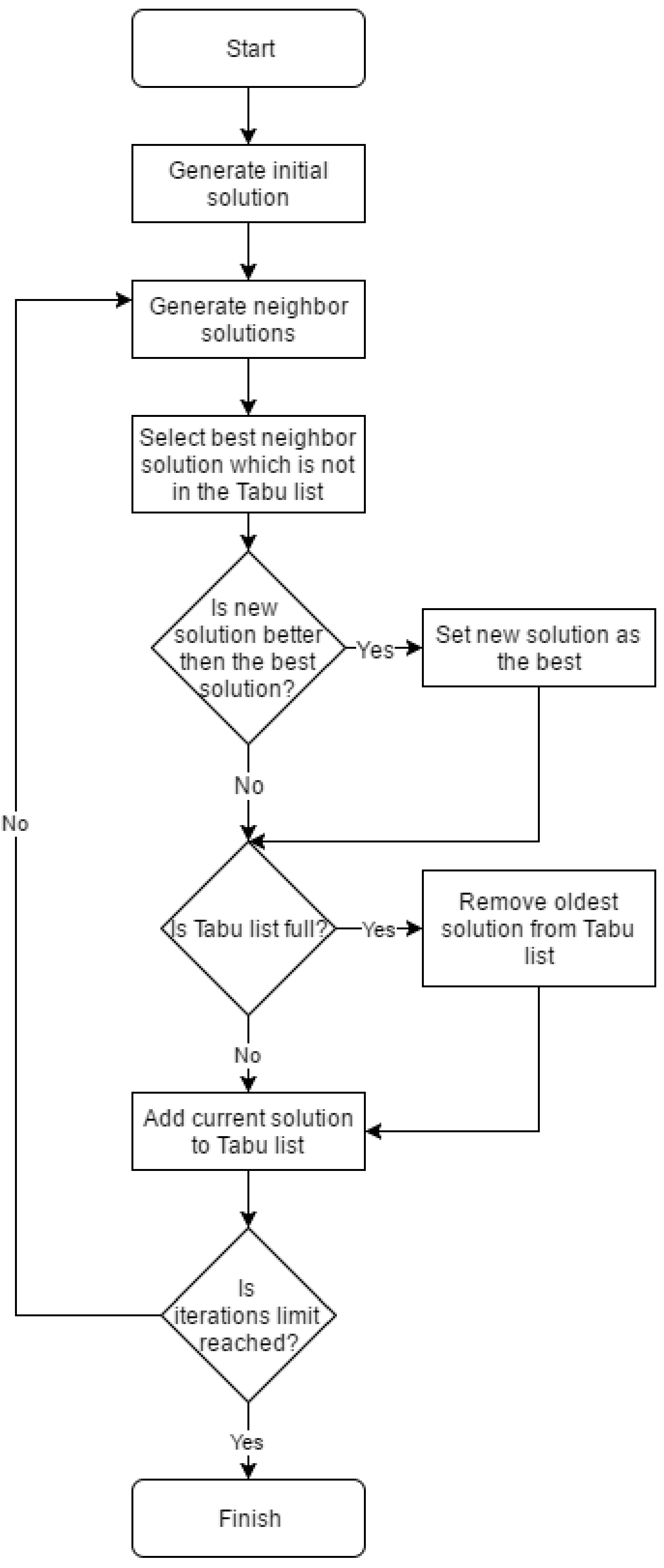

Start
Generate initial solution
Generate neighbor solutions
Select best neighbor solution which is not in the Tabu list
Is new solution better then the best solution?
Yes
Set new solution as the best
No
Is Tabu list full?
Yes
Remove oldest solution from Tabu list
No
Add current solution to Tabu list
Is iterations limit reached?
No
Yes
Finish

**Figure 1.5. General principle of Tabu Search algorithm**

Solutions stored in tabu list are forbidden for particular number of iterations. This is done to avoid repeating search state. In each iteration algorithm considers solutions, which are not in the tabu list. Then only one solution with smallest objective function value (not necessary better than current best solution) is selected as current best. Algorithm finishes after predetermined number of iterations [21].

Henn and Wäscher propose several approaches of tabu search for orders batching problem. First, authors suggest to use FCFS or CW2 for the initial solution generation. Then, three neighborhoods based on different moves (swap, shift, swap and shift) should be investigated.

Same authors introduced another approach to solve orders batching problem called Attribute-based Hill Climber (ABHC). This algorithm based on tabu search principle which could be described as follows: A set of solution attributes could be introduced. An attribute could be any feature of the solution. During local search phase, solution could be accepted only if it has smallest objective function value for at least one of its attributes. Using this algorithm three design choices should be made: initial solution, neighborhoods structure and attributes. For the initial solution and neighborhood structure, authors suggest to use same approach as for tabu search. For the attributes, they propose two sets: the first set characterizes each solution by pairs of customer orders which are assigned to the same batch. The second set of attributes is related to the assignment of orders to batches [23].

### 1.2.3 Population-based Approaches

The main difference between previously discussed algorithms and population based algorithms is that population based algorithms keep an array of candidate solutions rather than a single solution. All of the solutions in the array are used to obtain optimal solution. What prevents this approach from being just a parallel local search algorithm is that candidate solutions affect how other candidates will behave. This could happen either by good solutions causing poor solutions to be rejected and new ones created, or by causing them to be modified in the direction of the better solutions.

It may not be surprising that population-based algorithms use concepts from biology. One particularly popular set of techniques, collectively known as Evolutionary Algorithms (EA), borrows terms from population biology, genetics, and evolution. The most of the population based algorithms may be divided into generational algorithms, which update the entire array of solutions once per iteration, and steady-state algorithms, which update the few candidate solutions per iteration [21].

In 1999 Bullnheimer suggested a population based heuristic called Rank-based Ant System (RBAS), where each ant in the system represents a single solution. Henn modified this algorithm and applied it to orders batching problem. In this method, some number of ants are observed for predetermined number of iterations. The algorithm starts with a solution to put all orders to separate batches.

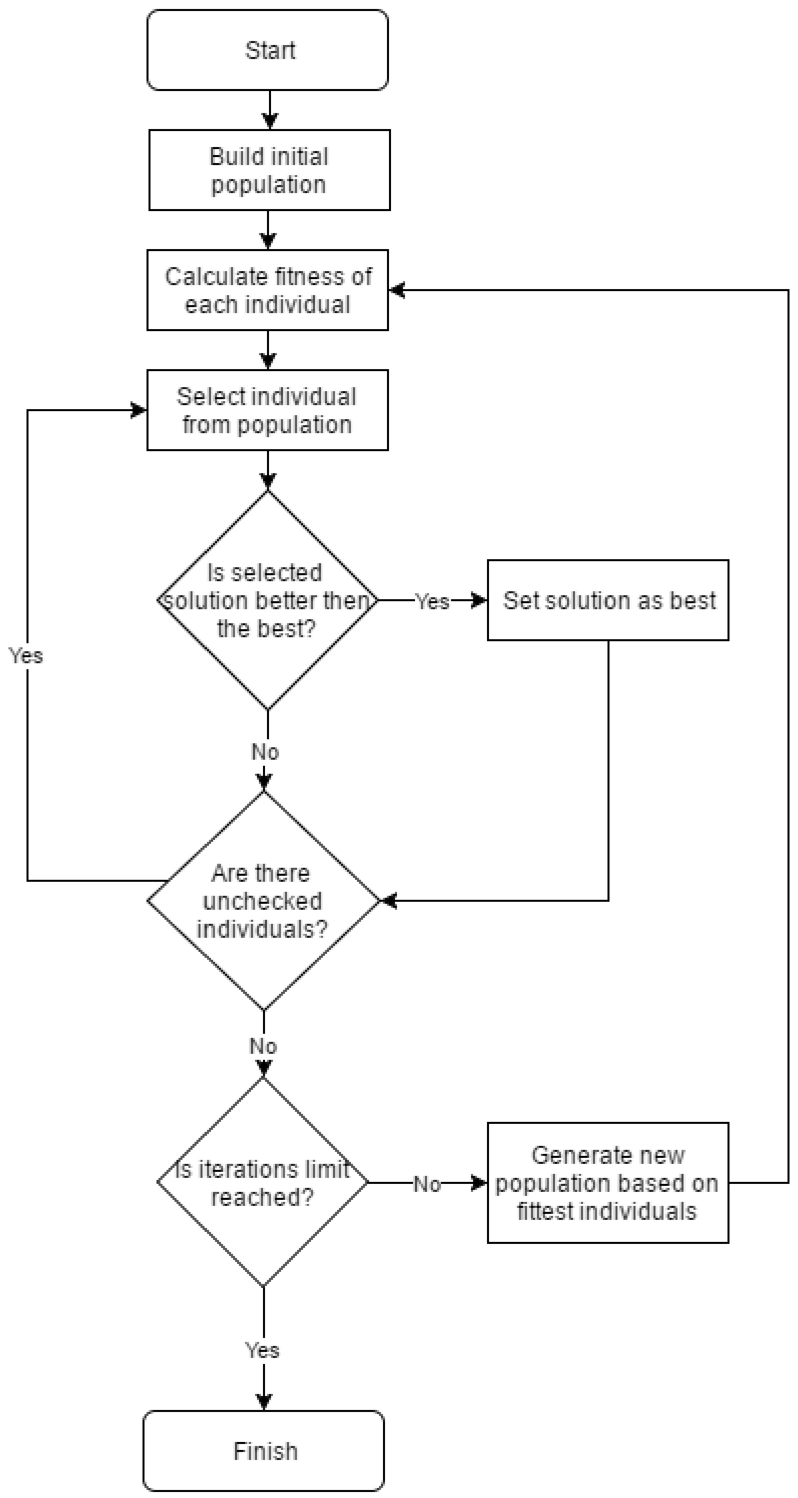

Start
Build initial population
Calculate fitness of each individual
Select individual from population
Is selected solution better then the best?
Yes
Set solution as best
No
Yes
Are there unchecked individuals?
No
Is iterations limit reached?
No
Generate new population based on fittest individuals
Yes
Finish

**Figure 1.6. General principle of population based algorithms**

In the next steps batches are combined as long as picking device capacity is not violated. After that, saving for each possible combination of two bathes are calculated. The better the saving the bigger pheromone intensity this batch has. Pheromone intensity of a batch pair is calculated as a sum of savings of all possible orders pairs combinations (one orders taken from one batch and other order is taken from other batch), divided by number of total possible orders pairs combinations between those two batches. Pheromone intensity defines probability which determines how it is likely that those two batches will be combined. When no new better solution could be found, local search is applied. This process is repeated for each ant. After that, pheromones of all order combinations are updated by “evaporating” poor solutions and adding additional pheromone to good solutions [11].

Another approach is to use Genetic Algorithms (GA). These algorithms generate large number of possible solutions and selects the best ones. The best solutions are modified (mutated) and combined (cross-over) in order to generate new solutions. Hsu used this approach for orders batching problem. Each solution is represented by a string of integer numbers, which groups orders into batches. The fitness of the solution is calculated as difference between length of longest tour in population and its tour length. So, the smaller the length of tour, the more fit is the solution, and it is more likely that this solution will be used to generated new solutions [24].

## 1.3 Performance of Algorithms

In order picking systems in general, the service level depends on order delivery time, order integrity and accuracy [5]. Order delivery time is closely related with the order picking time, since picking time is integral part of delivery time. So, optimization of picking process is vital for every warehouse. According to Niuwenhuyse and de Koster order picking time consists of [25]:

- Travel time. Time spent in traveling across the warehouse.
- Search time. Time required to find items.
- Pick time. Time needed to move item to picking device.
- Setup time. Time needed for setup tasks at the beginning of picking operation.

As pointed out by Tompkins et al., half of the order processing time is spent while travelling [3]. Tompkins also states that travel time is the main measure of picking process performance, because other times are constant or could be ignored because they are low.

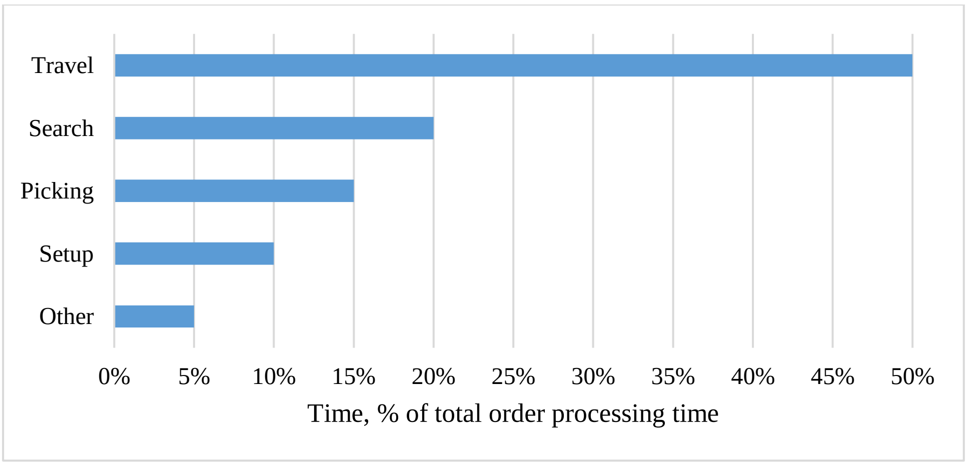


**Figure 1.7. Distribution of time needed to process the order [3]**

In modern warehouses where advanced search systems are used, the impact of travel time is even greater. Assuming that traveling speed is constant, minimization of travel time is same as minimization of total travel distance needed to collect all orders. So, the main measure which will be used for comparison of solutions in this work, will be the total travel distance.

There are plenty of numerical experiments done in order to check how well one or the other algorithm performs. However, there is no in-depth analysis on how those algorithms compares to each other on the same warehouse layout and on the same conditions. Published results are usually based on some very specific settings of capacity of picking device, warehouse size and layout, number of customer orders, demand structure and other. Therefore, it is hard to conclude one method's superiority over the other.

Earliest analysis has been done on simple heuristics like priority-based and seed algorithms. Gibson and Sharp analyzed how well priority based algorithm using space filling curve compares to random seed algorithm with different orders congruency rules. They used warehouse with 800 locations. The experiments showed that seed algorithm which uses aisle metrics as its distance parameters provides the shortest travel time [26].

Rosenwein had analyzed the performance of seed heuristic, using different orders congruency rules. He used Smallest Center of Gravity, Smallest Sum and Smallest Number of Additional Picking Aisles rules. There was warehouse with 750 storage locations used in his experiments. In this particular case, Smallest Number of Additional Picking Aisles rule generated shortest tour lengths [27].

Later de Koster and others performed experiments on both seed algorithms and savings algorithms. The experiments were made on different warehouse sizes. They used warehouses with 240, 400 and 1250 storage locations. Another variable used in these experiments was the capacity of picking device. These experiments showed that for seed algorithms cumulative mode outperforms the single mode. Also, the best results were obtained using seed algorithms which use Largest Number of Picking Aisles, Longest Travel Time and Largest Aisles Range rules for the seed selection. Performance of seed heuristics were compared with the performance of FCFS heuristic. The results showed that seed algorithm generated up to 19% shorter tour length on small warehouse and up to 7.5% shorter tour length on large warehouse [5].

Same authors conducted another experiment, which was concentrated on CW2 heuristic research. The experiment showed that on small warehouse this heuristic performs better than the best seed heuristic. When compared to FCFS rule, CW2 on average generated 20% shorter travel distances. However, in larger warehouse, seed algorithm generates better solution. Also, CW2 consumes up to 200 times more computing power than seed algorithm [5].

More detailed experiments on seed algorithms were made by Ho and others. They compared how different seed rules and different order congruency rules affects the tour length. There was a warehouse with 384 storage locations used in these experiments. The experiments showed that combination of Smallest Number of Picking Aisled rule for seed selection and Smallest Number of Additional Picking Aisles rule for order congruency generates the best results. Later the same authors conducted that these results can be improved by changing orders congruency rule to Shortest Average Mutual-nearest-Aisles Distance rule [19].

Henn and others compared ILS, RBAS and CW2 algorithms against FCFS. In this experiment, they used warehouse with 900 storage locations. Experiments showed that CW2 generates 17% shorter tours compared to FCFS. Both, ILS and RBAS, generated 20% shorter travel distances than FCFS. Difference between ILS and RBAS results was smaller than 1% [11].

Henn and Wäscher made numerical experiments, where they compared Tabu Search and ABHC against CW2 algorithm. In this experiment, they used warehouse with 900 storage locations. The results showed that Tabu Search generates 4% and ABHC generates up to 5% better solutions than CW2 [23].

Albareda-Sambola researched how well VLS algorithm performs against seed, savings and FCFS algorithms in settings described by de Koster and Ho. in their experiments. Their results showed that on average VLS generates better solutions than seed or savings algorithms. VLS was able to found better solutions in most cases and on average generated 19% better solutions than FCFS heuristic [22].

Based on research of other authors two benchmark algorithms are selected. The first one is FCFS heuristic. This algorithm will represent the lowest allowed performance. Another algorithm, which will be used as desired performance benchmark is CW2. This algorithm was well tested in various environments and was used in many numerical experiments. This makes CW2 a good benchmark, because it allows to compare performance of new algorithm to other heuristics, which was compared against CW2.

# 2. Methodology

As noted before, the aim of this work is to develop an algorithm which would help to group customer orders. The desired output of the algorithm is the groups (batches) of orders which need lowest travel distance to pick all needed items, considering picker's capacity limitations.

However, to test the performance of new algorithm, it needs to be compared to other algorithms in the exact same conditions. The simulated warehouse environment model is used for this purpose. This model describes the layout of the warehouse, picker's movement, its capacity and orders structure. To be able to use results of this experiment and compare the performance of algorithms to other algorithms in other experiments, the testing environment have to be as similar as possible to environments used in experiments done by other authors. Having this in mind, the main features of simulated warehouse environment are selected: layout of the warehouse (rows, items per row), number of orders, maximum number of items per order, capacity of the picker. All these features are used as the variables in the numerical experiments. This let evaluate the performance of the algorithm in more than one features environment and identify what parameters makes the new algorithm perform worse or better, to optimize the parameters of the new algorithm and to test it against other algorithms in same conditions.

The algorithm which will be developed is a population based algorithm and is often called Genetic Algorithm (GA). This algorithm borrows main ideas from genetics and evolution sciences. The main idea of this algorithm is that individuals of population can evolve to fit their function better. The evolution of the individuals is created by breeding and mutations. Fitter individuals get better probability of having off-springs, which leads to fitter and fitter individuals in each generation.

Based on literature two algorithms are selected to serve as benchmarks for a new algorithm. The first one is First Come First Served (FCFS) algorithm. This algorithm will represent the lowest allowed performance of the new algorithm. FCFS algorithm is also very popular in practice and is widely used in warehouses because of its simplicity. However, the results of this algorithm are random due to its nature.

The other selected algorithm is the second adaptation of Clarke and Wright (CW2) algorithm for orders batching problem. This algorithm will represent the desired performance of the new algorithm.

All three algorithms will be compared in multiple warehouse environments. To measure and to compare performance of the algorithms, tour distance and computation time will be used as performance indexes. The desired result of the GA algorithm is to generate same or better results than CW2 algorithm and to be calculated in reasonable amount of time. This would make GA algorithm applicable to real world warehouse environments.

Also, the longest GA performances will be analyzed in order to determine, how good are the algorithm parameters, and to find insights, how the performance of the algorithm could be improved.

## 2.1 Parameters of Simulated Warehouse Environment

In order to be able to compare results of this experiment to results of experiments done by other authors, the environment of the experiments should be as similar as possible. As mentioned before, there was identified the key features of the warehouse environment:

1. Warehouse layout
2. Orders structure
3. Capacity of the picker

### 2.1.1 Warehouse Layout

The most of the numerical experiments done in orders batching topic used the single block warehouse layout. This layout is presenter in Figure 2.1.

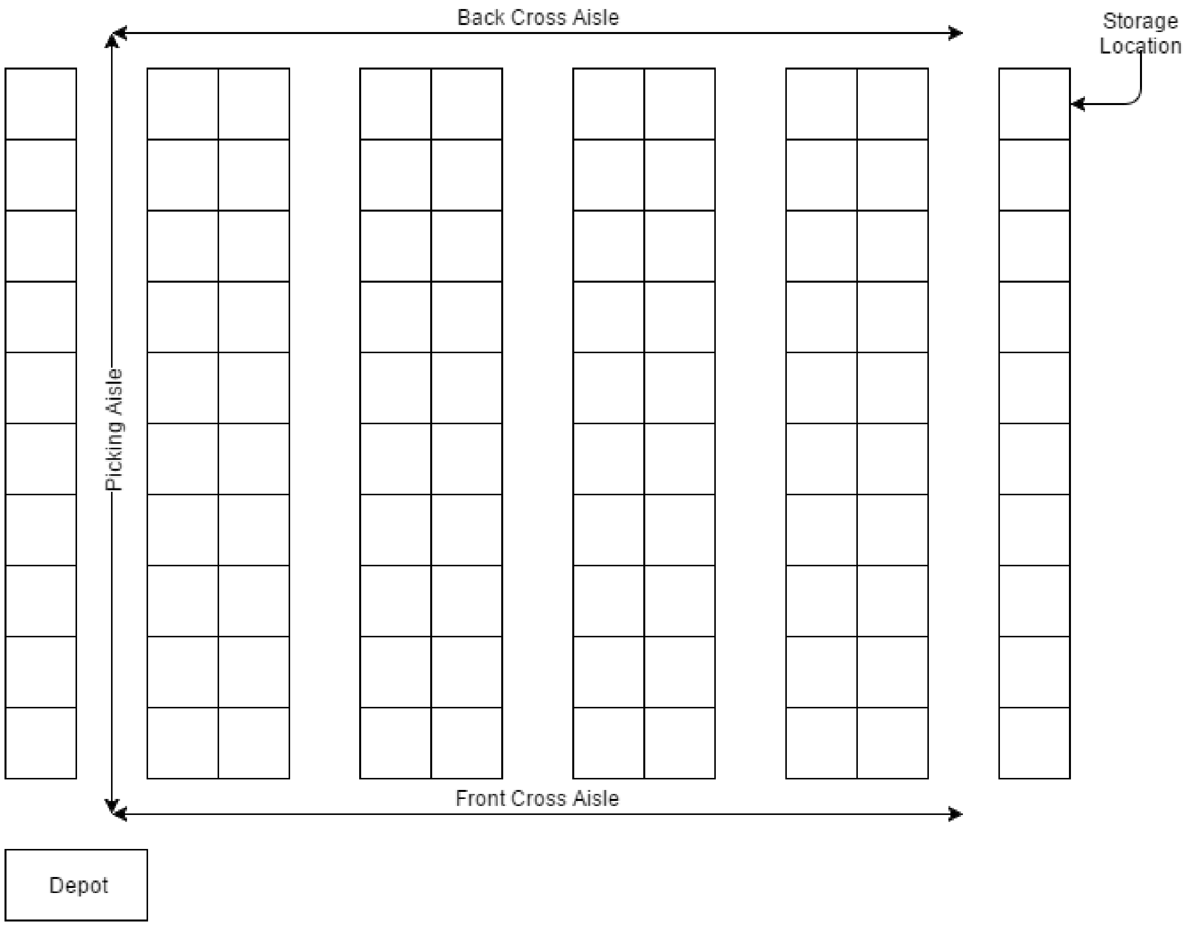


**Figure 2.1. Single block warehouse layout**

In this, simplified warehouse layout, all items storage places are equal in size. All items are stored in parallel lines on both sides of picking (vertical) aisles. All picking aisles are same length. Single block layout has two cross aisles, one in the front and one in the back of warehouse. The depot is the place in the warehouse where picking tour starts and ends. All picked items are stored in depot.

The aim of the experiment is to compare performance of the algorithms in same conditions and not the tour length itself. Because of this and for the sake of calculations simplicity, no standard distance measurement units are used in this part of the experiment. Rather, relative measurement units are used. It is determined that one storage location takes 1x1 relative measurement units.

The length of picking aisle depends on the number of items in that aisle and is equal to number of items in one storage line. It needs 3 relative distance units to travel from one picking aisle to another, so the length of cross aisle is:

$$3(n - m)$$

Where: n is the number of last picking aisle, and m is the number of first picking aisle.

The variables of the warehouse layout are the number of rows and number of items in one row. These variables represent the size of the warehouse. There are nine different combinations of rows and items per row used representing small, medium size and large warehouses.

1. Small warehouse:
    a. 10 rows, 10 items per row;
    b. 10 rows, 20 items per row;
    c. 10 rows, 30 items per row.
2. Medium size warehouse:
    a. 20 rows, 20 items per row;
    b. 20 rows, 30 items per row;
    c. 20 rows, 40 items per row.
3. Large warehouse:
    a. 30 rows, 30 items per row;
    b. 30 rows, 40 items per row;
    c. 30 rows, 50 items per row.

In real life warehouses items are usually stored in the warehouse in the way, that the most required items are stored as close to the depot as possible. However, in this numerical experiment items are stored randomly for the sake of computations simplicity.

### 2.1.2 Orders Structure

Orders structure is described by two variables: number of orders and maximum items per order. These two variables define how labor intensive is orders picking process. The more orders and more items per order means more work for the picker.

Values for the number of orders are: 5, 10, 20 and 30. Number of orders shows, how many orders are used to create the batches. Number of orders also represents the demand for the items stored in warehouse.

Values for the maximum number of items per order are: 5 and 10. This parameter with the picker's capacity parameter describes how big the orders are.

### 2.1.3 Picker's Capacity

This parameter shows how much items, picker can transport at once. This also could be understood as the size of one item. Therefore, the bigger capacity means bigger picking device or smaller items in the storage. The values of picker's capacity are: 10, 15 and 20. Again, the ratio of picker's capacity and maximum number of orders shows how big orders are. The bigger the orders, the more picker has to travel, due to frequent need to come back to the depot.

## 2.2 Genetic Algorithm

Genetic algorithms (GA) borrow ideas from genetics and evolution. The main idea behind this algorithm is that big enough population can produce good solutions using reproduction and mutation mechanisms.

First it is necessary to understand what population, individual, fitness and other terms of GA means in context of order batching problem. The solution to orders batching problem is orders batches that minimizes travel distance and does not violate picker's capacity limits. Therefore, in terms of GA, the individual is one of possible solutions, in other words - a group of batches, which does not violate the capacity constraint. The fitness of this individual then could be described by travel distance needed to pick all items in it. The less picker has to travel to gather all items in the solution, the fitter the solution is. A group of individual solutions is then called a population.

Second, GA has several parameters that should be tweaked, so algorithm could work fast and also generate good enough solutions. These are:

1. Size of population
2. Number of generations
3. Mutation ratio

The main processes of the Genetic algorithm are:

1. Generation of initial population
2. Determination of fitness of each individual in the population
3. Reproduction and creation of new population

### 2.2.1 Initial Population

Genetic algorithm is the optimization algorithm. Therefore, it cannot create solution having only the problem statement and conditions. Genetic algorithm needs initial population as an input to have something to start with. Then, with each generation it can create a new population and new solutions for the problem.

In this case, solutions for the initial population is created using FCFS algorithm. First the initial orders are generated. Orders then are shuffled and FCFS is used to generate order batches. The result of the FCFS is one individual. Then initial orders are shuffled again, and FCFS creates another individual. This process is repeated as long as there are still missing individuals in the population. The size of the population is described later chapter.

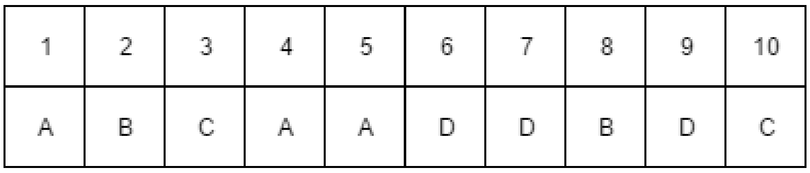
### 2.2.2 Creation of a New Population

New populations are created by combining individuals from previous generation and individuals created by cross-over mechanism. The ratio of how much old individuals compared to how much new individuals should be in the new population is another parameter that could be changed to optimize the algorithm. In this specific case, it is determined that new population should be created by 50% of old individuals and 50% of new individuals.

Which old individuals is used in new population is determined by their fitness. The better the fitness, in other words, the smaller the distance the solution generates, the more likely, that this individual will be in the next generation. Therefore, 50% of fittest individuals are selected and added to the new population. Other individuals required for the population are created using these fittest individuals via cross-over process. The cross-over stage is described in later chapter.

### 2.2.3 Genome

Each individual is represented by its genome. In context of orders batching, genome is just encoded orders batches. Genome is constructed of individual genes. One gene represents one specific order and batch ID. For example, if there is an individual which has 10 orders, and these orders are grouped to 4 batches, the genome of such individual would look like Figure 2.2:



| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
|---|---|---|---|---|---|---|---|---|---|
| A | B | C | A | A | D | D | B | D | C |

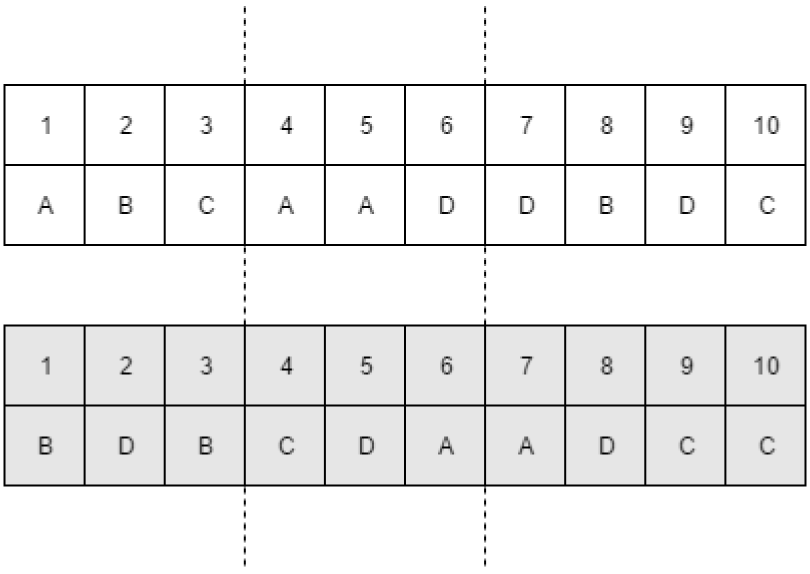
**Figure 2.2. Genome of the individual**

The first line of Figure 2.2 represents the orders. There are 10 different orders. The bottom line represents the batch in which each order is. For example, order 1 is in batch A. There are also orders 4 and 5 in the same batch. Batches assignment to orders are the main difference between each individual solution. Each solution has its own unique genome.

### 2.2.4 Cross-over

Cross-over (sometimes called breeding) is the process which creates new solutions for the populations using previous solutions. There are many different ways to implement cross-over functionality mentioned in the literature. In this particular case, cross-over is done by combining genes of two individuals. Therefore, first 50% of fittest individuals in current population are selected to mating pool. Then those individuals are paired by their fitness. There are two new individuals (off-springs) created from each pair of old individuals (parents).

The combination of genes of parents is a stochastic process. First, two random numbers are generated. Each of these number can have a value from 0 to the size of the genome. These numbers represent the genome split places. For example, if there are two parent individuals, with ten orders each, and random number which represents split locations in genome are 3 and 6, then such split would look like Figure 2.3:



| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
|---|---|---|---|---|---|---|---|---|---|
| A | B | C | A | A | D | D | B | D | C |

| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
|---|---|---|---|---|---|---|---|---|---|
| B | D | B | C | D | A | A | D | C | C |

**Figure 2.3. Split of two parent individuals**

As showed in Figure 2.3, genome of the parents is split into three parts. The middle parts are then exchanged; therefore, two new individuals are created. The end result is shown in Figure 2.4.

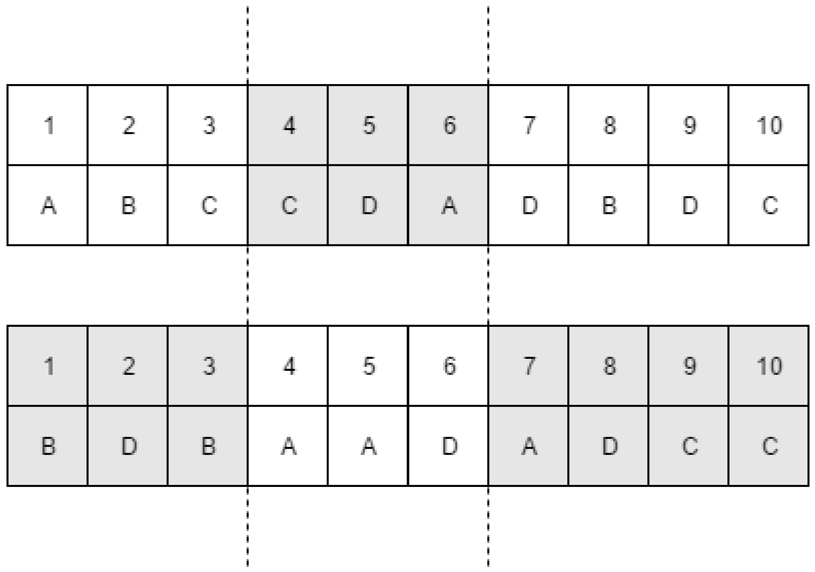


**Figure 2.4. New individuals created from parents' genes**

As shown in Figure 2.4, first new item contains genes 1-3 and 7-10 from first parent and genes 4-6 from second parent, while other offspring contains genes 1-3 and 7-10 from second parents and genes 4-6 from first parent.

Any changes in genes could lead to the picker's capacity constraint violations. When such cases happen, genome repair should be done. To repair the genome, three options are considered. First, the algorithm tries to move the new item that violates capacity, to the other batch by applying the Best-Fit rule. Order is included to another batch if it fills all available empty place in that batch. If there are no batches, where order could be included by Best-Fit rule, algorithm tries to move this order to another batch by applying Next-Fit rule. Algorithm tries to find batch, where this order would just fit and not violate capacity constraints. If no batches were found, new batch is opened and order is placed in it.

### 2.2.5 Mutation

Mutation is a process, when two orders in the same solution swap their batches. Mutation is needed to avoid or at least to minimize the local optima situations. Mutation is done immediately after the creation of new individuals and before the correction of the genome. Mutation is a stochastic process, there are two random orders selected in the solution and their batches are swapped. The mutation process is presented in Figure 2.5.

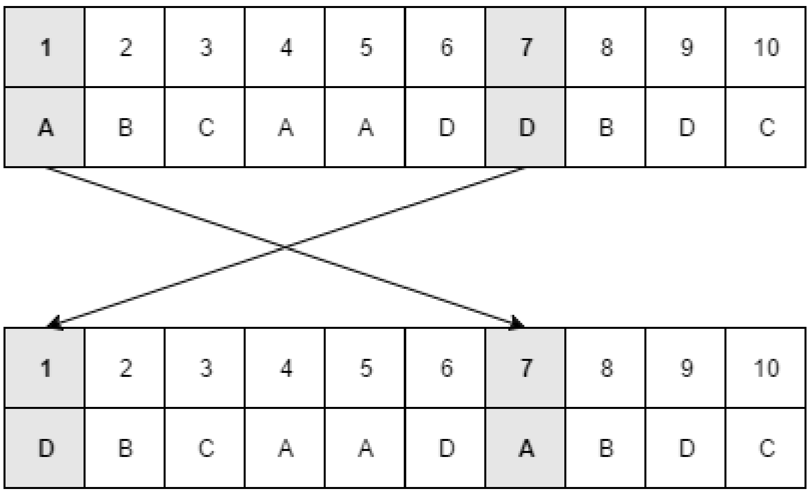


**Figure 2.5. Mutation of the individual**

The rate of the mutation represents the probability of the individual to mutate. To find out, what rate of the mutation gives the best results, additional experiments were made. The results of these experiments are shown in  Figure 2.6.

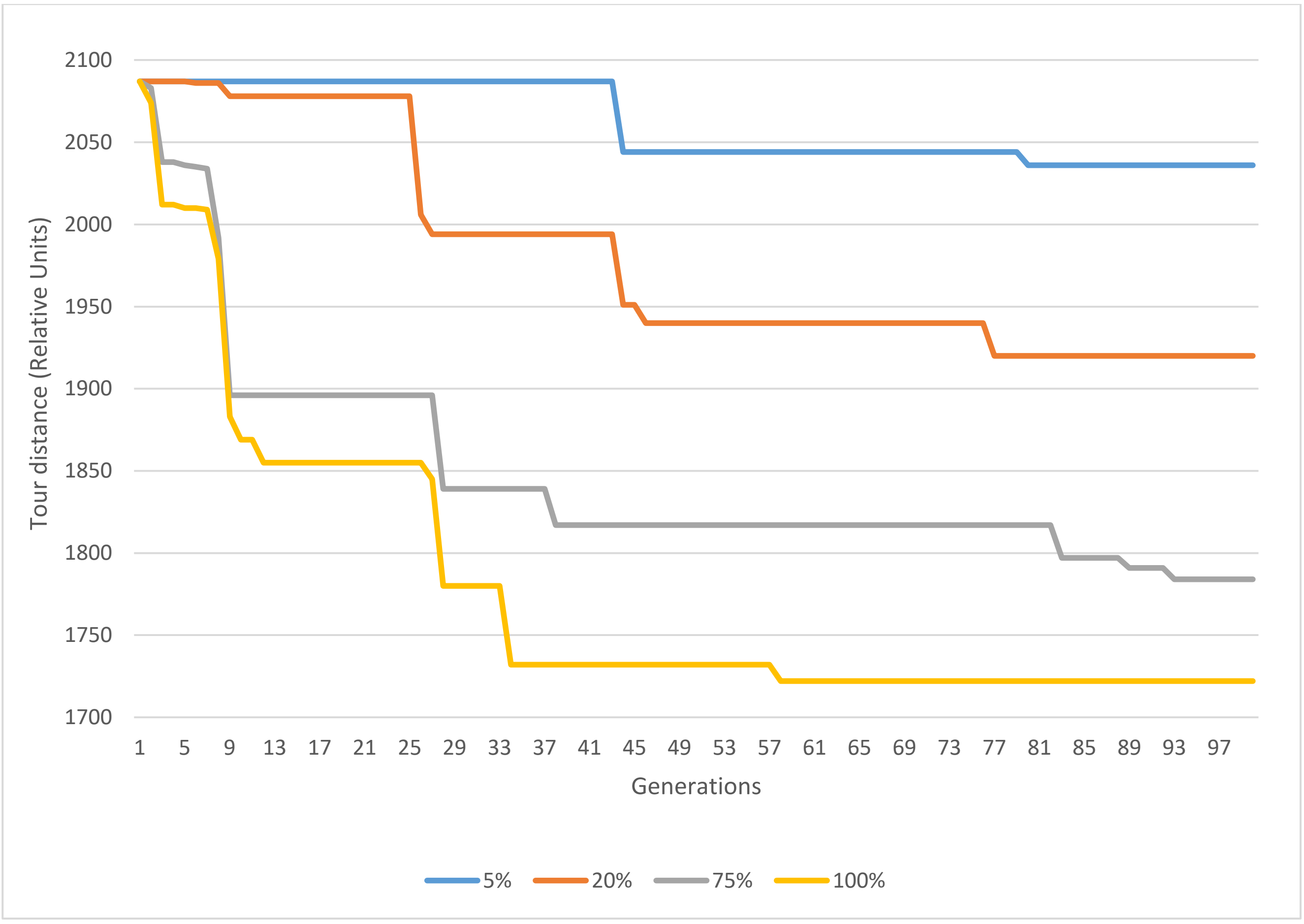


**Figure 2.6. Mutation probability and travel distance**

There were four experiments conducted in order to determine what mutation probability generates best results. All experiments were done on same conditions, only mutation probability was changed. Tested probabilities of the mutation are: 5%, 20% 75% and 100%. The mutation probability shows how often mutation should occur after the new individual is created. Therefore 5% mutation probability means that mutation process will be applied to 5% of new individuals. Each experiment has same starting orders, same initial population and population size and are terminated after 100 generations.

To be able to judge the mutation probability influence on solution generation, there was cross-over process disabled. It is needed because cross-over is stochastic process, so even having same conditions it still can generate different results. To prevent that, cross-over was disabled, and only mutation was used to create new populations.

As seen in the Figure 2.6, when mutation probability is 5%, GA was able to generate solution, which has orders batches, where items would be collected in 2036 relative distance units. As seen from the graph, new best solutions were found just two times: at generation 43 and at generation 80.

When mutation probability was increased to 20%, algorithm was able to generate solution which needs 1920 relative units to pick all items in it. Mutations was applied more often so algorithm was able to found more solutions that are better than previous best. New best solutions were found in generations: 6, 9, 26, 27, 44, 46 and 78. GA with 20% mutation probability was able to overcome best solution of GA with 5% mutation probability in just 26 generations.

Next, algorithms with mutation probability of 75% was tested, therefore 3 out of 4 individuals was mutated in each generation. This time algorithm was able to create solution which would need 1784 relative distance units to pick all items in it. There was a gap between generations 38 and 83, when algorithm was not able to get any better solutions. In all other times, it made solution better and better with almost every generation. GA with 75% mutation probability was able to overcome the best solution from GA with 20% mutation probability in just 9 generations.

The best results (the lowest travel distance) is generated when mutation probability is very high – 100%, meaning that every new individual should be mutated. GA with 100% mutation probability created solution which needs 1722 relative distance units to pick all items. It is interesting, that it reached its best solution in generation 58 and after that is was not able to improve, suggesting that this is best possible or near best possible solution. GA with 100% mutation probability overcome the best result from GA with 75% mutation probability in 28 generations.

The need for high mutation probability in GA can indicate that population lacks differentiation in individuals, and mutation gives exactly that, it does not let come to the local optima, when no new better solution can be found just by cross-over operation. Therefore, based on these results, selected mutation probability is 100%.

### 2.2.6 Size of Population

Population size relates to the quality of the solution. The bigger population means more differences in individuals, therefore, more chances to create better and better new individuals. However, the bigger population also needs more time to process and calculate. Also, when there are just a few orders that have to batched, there are no need for a big population, because individuals just start to repeat, but in cases, when there are 30 or more orders to collect, big population is necessary, because small number of individuals may not converge to a good solution. Having this in mind, it is determined that population should be directly related to the total number of orders. Therefore, six experiments were conducted to determine what populations size should be used. Considered dependencies are shown in Table 2.1. Where a is the number of orders and n is the size of population. All experiments where conducted on the same conditions using 600 storage locations warehouse layout, 40 orders, 10 maximum items per order and 10 items pickers capacity. Experiments were terminated after 50 generations. The results of these experiments are shown in Table 2.1.

**Table 2.1. Travel distance and computation time dependency on population size**

| Dependency (n – population size, a – number of orders) | Distance (Relative Units) | Computation time (s) |
|---|---|---|
| $n = \frac{a}{2}$ | 4602 | 3 |
| $n = a$ | 4572 | 11 |
| $n = 2a$ | 4524 | 19 |
| $n = a^2$ | 4446 | 54 |
| $n = 2a^2$ | 4383 | 97 |
| $n = a^3$ | 4378 | 203 |

The fastest, however the worst by the means of travel distance is the configuration where populations size is equal to the half of the number of orders. The best in travel distance, but the longest one to compute was the configuration which uses number of order cubed. Therefore, the selected dependency for the size of the population is: $n = 2a^2$. The result of this configuration is very close to solution of the best solution, however it needed two times less time to finish its calculations.

### 2.2.7 Number of Generations

Similarly, to the population size, number of generations also relates to the quality of the solution. Number of generations determines how much iterations should be made and when should the algorithm terminate. Small number of generations completes fast; however, the solution then is far from optimal. Large number of generations, lets to create good solution, however it takes a lot of time to complete. Also, when optimal solution is reached in early generations, all other iterations are just a waste of resources. It is determined that the termination of the algorithm should not depend on a static number. Every situation is individual, therefore same number of generations could make the algorithm to complete too early in one conditions and to do redundant calculations in others. Thus, it is determined to make the number generations, which establishes when algorithm should terminate should depend on how old the current best solution is.

To find out, what is the optimal termination parameter, five experiments where conducted. All experiments were made on the same conditions using warehouse layout with 600 storage locations, 40 orders, 10 maximum items per order and 10 items picker can carry at the same time. The main difference between experiments is the termination parameter. It is described by the age of the current best solution. The age of the best solution is measured in number of generations, the algorithm was not able to find a better solution. Five different termination parameters are checked in this experiment: 10, 50, 100, 200 and 300 generations.

The results of these experiments are shown in Table 2.2. They show that when allowed age of the best solution is low the result is found faster. However, this result is not as good as in cases when allowed age of the best solution is higher. Therefore, the best solution in this particular case, looks to be the 100 generations. It produces very similar result to the results received by using 200 and 300 generations as the limit of the best solution age, however algorithm terminated a lot earlier. Therefore 100 static generations will be used as the termination parameter of the Genetic algorithm. This lets to complete calculations fast if good solution is found early. Also, it prevents premature solution selection.

**Table 2.2. Results of generations experiments**

| Age of best solution (Number of generations) | 10 | 50 | 100 | 200 | 300 |
|---|---|---|---|---|---|
| Number of generations needed to complete calculations | 26 | 137 | 288 | 511 | 524 |
| Travel distance (Relative Units) | 4922 | 4759 | 4701 | 4699 | 4699 |

# 3. Results

As mentioned before, the experiments variables are:

1. 9 different warehouse layouts (rows and items per row);
2. 4 different numbers of orders
3. 3 different max items per order and capacity combinations

Therefore, it was 108 different numerical experiments conducted in total. All results are divided into three parts representing different warehouse sizes: small, medium and large. The key performance measurements are the tour length savings (%) compared to the FCFS algorithm and the solution computation time (s). First, tour distances for each algorithm are calculated. Then savings for both, CW2 and GA algorithms are calculated and represented as how much distance it is saved compared to the FCFS algorithm. These savings are then used to compare performance of both CW2 and GA algorithms. Computation time is used to determine how well the algorithm could be adapted to real warehouse situations. It is determined that unacceptable time is 5 minutes. As mentioned before, the desired performance of GA should be same or better than CW2. All computations are done on a laptop having intel i5 four core processor and 8GB of ram. However only one core of the processor is used for the computation to make it easier to compare the results.

## 3.1 Small Warehouse

Small warehouse is represented by tree different layouts, having 100, 200 and 300 storage locations respectively. There were 12 experiments conducted for each layout.

### 3.1.1 Warehouse with 100 Storage Locations

Savings of algorithms for warehouse with 100 storage locations is represented in Table 3.1.

**Table 3.1. Results of small warehouse with 100 storage locations**

| Capacity | Max items per order | Orders | Savings CW2 | Savings GA | Difference (GA – CW2) |
|---|---|---|---|---|---|
| 20 | 10 | 30 | 15% | 15% | 0% |
| 20 | 10 | 20 | 11% | 11% | 0% |
| 20 | 10 | 10 | 8% | 8% | 0% |
| 20 | 10 | 5 | 14% | 14% | 0% |
| 15 | 5 | 30 | 32% | 34% | 2% |
| 15 | 5 | 20 | 23% | 29% | 6% |
| 15 | 5 | 10 | 16% | 19% | 3% |
| 15 | 5 | 5 | 19% | 19% | 0% |
| 10 | 5 | 30 | 32% | 37% | 5% |
| 10 | 5 | 20 | 23% | 29% | 6% |
| 10 | 5 | 10 | 16% | 17% | 1% |
| 10 | 5 | 5 | 19% | 19% | 0% |

As it is show in Table 3.1, in warehouse with this specific layout GA generates same or better results than CW2 algorithm.

CW2 savings ranges from 8% to 32% when compared to the FCFS algorithm. Average savings for CW2 is 19%, which is very similar to the results, other authors in their experiments was able to achieve. Results in Table 3**.1** show that in small warehouse with 100 storage locations CW2 performs worst, when parameters are: 10 orders, picker's capacity is 20 and maximum items per one order is 10. In this case CW2 generates just 8% better results than FCFS. This is odd result, because it is in the middle of the variable space. CW2 generates better results in both cases, when there are more orders and when there are less orders. It is most likely that, at the beginning of that particular experiment initial orders was randomly created in the way, that FCFS generates pretty good solution using these orders. In that way both, CW2 and GA, would not have much room to improve the solution.

CW2 performs best with parameters: 30 orders, pickers capacity is 10 or 15 and maximum number of items per order is 5.

GA results ranges from 8% to 37% when compared to FCFS. On average GA creates 21% better results than FCFS and 2% better results than CW2 algorithm. Similar to CW2, GA performs worst when parameters are: 10 orders, picker's capacity is 20 and maximum number of items per order is 10. Genetic algorithm creates best result when parameters are: 30 orders, picker's capacity is 10 and maximum items per order is 5.

Looking further, it can be noticed that both algorithms performs worst when picker's capacity is 20. For both, CW2 and GA algorithms, their four worst results are achieved when picker's capacity is 20. This all correct, when the nature of the orders batching problem is analyzed a bit deeper.

A good solution for this problem would have to make the utilization of picker's capacity as good as possible and also would prevent the redundant travel from depot to picking are and vice versa. The situations, when experiment parameters have big capacity, small number of orders and small number of items per order, should have better results by themselves, when compared to the experiments having small capacity, big number of items per order and big number of orders. The bigger capacity means, that picker can carry more items. Also, this means that picker has to travel to the depot less often. Therefore, experiments having relatively big, picker's capacities, small number of orders and small number of items per order, would not leave much room for the improvement. This is exactly what results in Table 3**.1** show.

The bigger the picker capacity, the smaller amount of orders and less items, order can have, leads to situations when all improvements can be done just by means of the item picking sequence, because most of the orders can be picked at the same tour. And both, CW2 and GA, are good at improving the capacity utilization and reducing number of necessary tours across the warehouse.

Therefore, algorithms have not much room to improve the solution, however, they are still able to improve it by 8% even, when experiment is the worst-case scenario for both algorithms.

The computation time for each experiment is shown in Table 3.2.

**Table 3.2. Computation times for warehouse with 100 storage locations**

| Capacity | Max items per order | Orders | FCFS computation (ms) | CW2 computation (ms) | GA computation (ms) |
|---|---|---|---|---|---|
| 20 | 10 | 30 | 0 | 60 | 69313 |
| 10 | 5 | 30 | 0 | 47 | 38457 |
| 15 | 5 | 30 | 0 | 47 | 30955 |
| 20 | 10 | 20 | 0 | 15 | 19403 |
| 10 | 5 | 20 | 0 | 19 | 10668 |
| 15 | 5 | 20 | 0 | 10 | 10273 |
| 20 | 10 | 10 | 0 | 2 | 3944 |
| 10 | 5 | 10 | 0 | 35 | 2320 |
| 15 | 5 | 10 | 0 | 2 | 1436 |
| 10 | 5 | 5 | 1 | 13 | 1017 |
| 20 | 10 | 5 | 0 | 0 | 601 |
| 15 | 5 | 5 | 0 | 0 | 313 |

Computation times in Table 3.2 are sorted by computation time in GA column from largest to smallest. It is clear, that GA needs significantly more time to compute its solution than both, FCFS and CW2. GA computation time ranges from 0.3s to 69s. The most important factor for the computation time is the number of orders. The more orders there are, the more time algorithm needs to complete. Also, computation time relates to the maximum number of items per order.

While GA needs much more time to calculate the solution, the amount of time it needs is still reasonable. 69 seconds is not much, when total tour distance could be 6% shorter compared to CW2 and up to 36% shorter compared to FCFS algorithm.

The performance of GA in warehouse with 100 storage locations is shown in Figure 3.1.

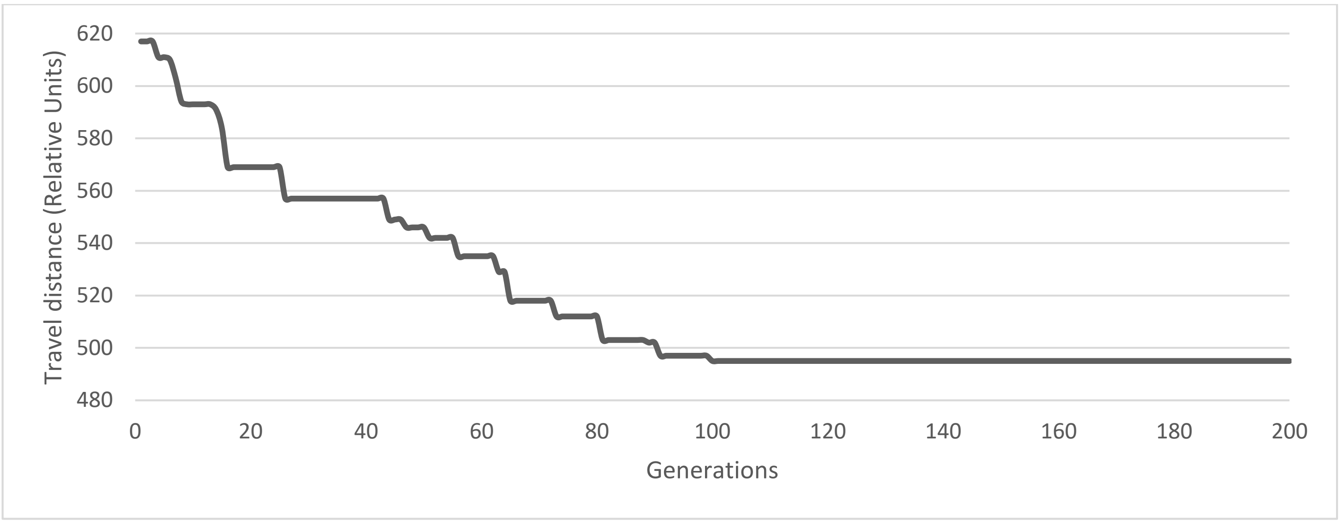


**Figure 3.1. GA performance in warehouse with 100 storage locations**

Figure 3.1 shows how best solution changes with each generation when parameters are: 30 orders, 5 maximum items per order and capacity is 10. Best solution at first generation is 617 relative units. The best solution is improved until generation 100. The best solution at that point is 495 relative units. The algorithm is not able to find better solution in next 100 generations, therefore it terminated at generation 200.

### 3.1.2 Warehouse with 200 Storage Locations

Savings results for warehouse with 200 storage locations are in Table 3.3.

**Table 3.3. Savings of warehouse with 200 storage locations**

| Capacity | Max items per order | Orders | Savings CW2 | Savings GA | Difference (GA – CW2) |
|---|---|---|---|---|---|
| 20 | 10 | 30 | 13% | 14% | 1% |
| 20 | 10 | 20 | 10% | 10% | 0% |
| 20 | 10 | 10 | 8% | 8% | 0% |
| 20 | 10 | 5 | 11% | 11% | 0% |
| 15 | 5 | 30 | 31% | 33% | 2% |
| 15 | 5 | 20 | 25% | 29% | 4% |
| 15 | 5 | 10 | 3% | 13% | 10% |
| 15 | 5 | 5 | 16% | 16% | 0% |
| 10 | 5 | 30 | 31% | 35% | 4% |
| 10 | 5 | 20 | 25% | 29% | 4% |
| 10 | 5 | 10 | 3% | 14% | 11% |
| 10 | 5 | 5 | 16% | 16% | 0% |

Again, GA performed as good or better then CW2 algorithm. CW2 was able to generate results from 3% up to 31% better then generated by FCFS and GA results ranges from 8% to 35%. On average CW2 and GA generated 16% and 19% better results than FCFS respectively. On average GA was better algorithm in terms of travel distance by 3%.

The most significant difference can be noticed when parameters of the experiment are: capacity is 10, maximum number of items per order is 5 and number of orders is 10. In this case CW2 generates just 3% better results than FCFS, while GA is able to improve the solution by 14%. Both algorithms perform lower than their average in this warehouse layout. Again, most likely random orders were generated in such way that FCFS is able to create good solution, therefore, both algorithms have little room to improve. Still GA was able to perform 11% better than CW2 in this particular case.

Both, CW2 and GA creates best solutions when parameters are: capacity is 10, maximum items per order is 5 and number of orders is 30. Relatively big number of orders and small capacity empowers both algorithms to use their ability to maximize utilization of picker's capacity, therefore to lower total travel distance.

GA performs worst in same conditions as in warehouse with 100 storage locations. Again, this happens because GA have no much room to improve the solution, due to specifics of the parameters, which are analyzed more detailed in previous chapter.

**Table 3.4. Computation time for warehouse with 200 storage locations**

| Capacity | Max items per order | Orders | FCFS computation (ms) | CW2 computation (ms) | GA computation (ms) |
|---|---|---|---|---|---|
| 20 | 10 | 30 | 0 | 54 | 67680 |
| 15 | 5 | 30 | 0 | 34 | 33433 |
| 10 | 5 | 30 | 0 | 36 | 29102 |
| 20 | 10 | 20 | 0 | 15 | 21379 |
| 10 | 5 | 20 | 0 | 9 | 11898 |
| 15 | 5 | 20 | 0 | 9 | 11532 |
| 20 | 10 | 10 | 0 | 2 | 4774 |
| 15 | 5 | 10 | 0 | 2 | 1442 |
| 10 | 5 | 10 | 0 | 1 | 1399 |
| 20 | 10 | 5 | 0 | 1 | 536 |
| 10 | 5 | 5 | 0 | 0 | 318 |
| 15 | 5 | 5 | 0 | 0 | 291 |

Computation time needed for solution computation is presented in Table 3.4**Klaida! Nerastas uorodos šaltinis.**. Computation time ranges from 0,3s to 68s. Computation results are very similar to the results from previous chapter, GA needs much more time to complete, however again, computation time is reasonable and GA could be used in real warehouse situations, therefore provide travel distance improvements as big as 11% when compared to CW2 and up to 35% shorter travel distance when compared to FCFS.

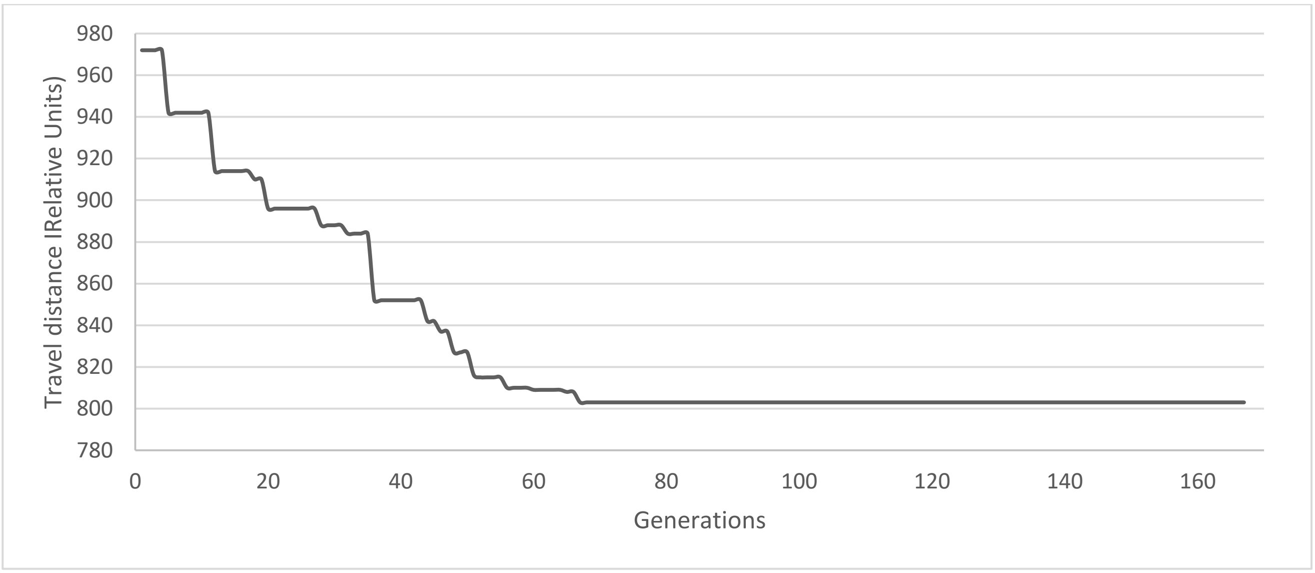


**Figure 3.2. GA performance in warehouse with 200 storage locations**

The change of best solutions in each generation for the best-case parameters is shown in Figure 3.2. In this example, GA was able to find best solution in 68 generations. GA started with the best solution which needs 972 relative unit to pick all items in it. Best solution after 68 generations needed 803 relative distance units to pick all items. After that, algorithm was not able to find better solutions for next 100 generations, therefore it terminated. In this case, GA took 29 seconds to finish.

### 3.1.3 Warehouse with 300 Storage Locations

**Table 3.5. Results of the warehouse with 300 storage locations**

| Capacity | Max items per order | Orders | Savings CW2 | Savings GA | Difference (GA – CW2) |
|---|---|---|---|---|---|
| 20 | 10 | 30 | 13% | 13% | 0% |
| 20 | 10 | 20 | 8% | 8% | 0% |
| 20 | 10 | 10 | 6% | 6% | 0% |
| 20 | 10 | 5 | 11% | 11% | 0% |
| 15 | 5 | 30 | 35% | 38% | 3% |
| 15 | 5 | 20 | 33% | 38% | 5% |
| 15 | 5 | 10 | 21% | 24% | 3% |
| 15 | 5 | 5 | 30% | 30% | 0% |
| 10 | 5 | 30 | 35% | 40% | 5% |
| 10 | 5 | 20 | 33% | 38% | 5% |
| 10 | 5 | 10 | 21% | 24% | 3% |
| 10 | 5 | 5 | 30% | 30% | 0% |

Results of the warehouse with 300 storage locations are represented in Table 3.5. Once again, results show that Genetic algorithm can be as good and better then CW2 algorithm in small warehouses. GA generated up to 5% and 2% on average better results than CW2.

CW2 was able to generate results ranging from 6% to 35%, while GA was able to create solutions which are from 6% to 40% better than results of FCFS algorithm. CW2 generated 23% shorter tour distances and GA generated 25% shorter distance on average comparing to FCFS. When compared with previous layouts, both algorithms improved results. This happens because, when warehouse gets bigger, total tour distance is more and more dependent on the items picking queue, rather than dependent on how much time picker needs to go to the depot. In cases, when relatively big capacity and small number of orders is used, algorithms still perform worst, however, when there is relatively small capacity big number of items to collect, algorithms improve solution more than in smaller layouts.

There are three cases, when GA outperforms CW2 by 5%. All of them have 20 or 30 orders, and 15 or 10 items picker's capacity. Which indicates, that when it comes to cases, when tour length depends more on items picking queue, rather than times picker needs to go to and from the depot, GA is able to create better results than CW2.

**Table 3.6. Computation time needed in warehouse with 300 storage locations**

| Capacity | Max items per order | Orders | FCFS computation (ms) | CW2 computation (ms) | GA computation (ms) |
|---|---|---|---|---|---|
| 20 | 10 | 30 | 0 | 51 | 71790 |
| 10 | 5 | 30 | 0 | 34 | 42593 |
| 15 | 5 | 30 | 2 | 45 | 31856 |
| 20 | 10 | 20 | 0 | 15 | 22466 |
| 15 | 5 | 20 | 0 | 20 | 14887 |
| 10 | 5 | 20 | 0 | 11 | 10954 |
| 20 | 10 | 10 | 0 | 2 | 3612 |
| 15 | 5 | 10 | 0 | 1 | 1411 |
| 10 | 5 | 10 | 0 | 1 | 1410 |
| 20 | 10 | 5 | 0 | 0 | 449 |
| 10 | 5 | 5 | 0 | 0 | 308 |
| 15 | 5 | 5 | 0 | 0 | 288 |

Computation times needed for the algorithms to complete are presented in Table 3.6. Once again, GA need much more time than CW2. GA computation time ranges from 0.3s to 72s. CW2 is able to get result in same situation from 1000 to 1500 times faster. However, 72 seconds in human scale is still reasonable computation time, moreover this computation can generate better results. These savings, even if they as small as few percent, adds up in long term, therefore makes order picking operation more efficient.

## 3.2 Medium Size Warehouses

Next, three different layouts representing medium size warehouses will be represented. The results of orders batching algorithms will be compared in each one of them and among them. Medium size warehouses are represented by layouts having 400, 600 and 800 storage locations.

### 3.2.1 Warehouse with 400 Storage Locations

Results for warehouse with 400 storage locations are presented in Table 3.7. Genetic algorithm again is able to create same or better results than CW2 algorithm. However overall results for both algorithms are not as good as results in warehouse with 300 storage locations.

CW2 algorithm was able to generate solutions which saves from 4% to 34% travel distance when compared to FCFS heuristic. The average CW2 savings on this particular layout is just 14%. Until now this is the worst average savings for this algorithm.

**Table 3.7. Algorithms results in warehouse with 400 storage locations**

| Capacity | Max items per order | Orders | Savings CW2 | Savings GA | Difference (GA – CW2) |
|---|---|---|---|---|---|
| 20 | 10 | 30 | 12% | 12% | 0% |
| 20 | 10 | 20 | 10% | 10% | 0% |
| 20 | 10 | 10 | 7% | 7% | 0% |
| 20 | 10 | 5 | 5% | 5% | 0% |
| 15 | 5 | 30 | 34% | 34% | 0% |
| 15 | 5 | 20 | 21% | 23% | 2% |
| 15 | 5 | 10 | 4% | 10% | 6% |
| 15 | 5 | 5 | 9% | 9% | 0% |
| 10 | 5 | 30 | 34% | 35% | 1% |
| 10 | 5 | 20 | 21% | 23% | 2% |
| 10 | 5 | 10 | 4% | 10% | 6% |
| 10 | 5 | 5 | 9% | 9% | 0% |

Genetic algorithm generated solutions ranging in savings from 5% to 35% when compared to FCFS. Average saving of GA in this layout is 16%. Same as for CW2, the average result is worse comparing to small warehouse layouts.

GA created 1% better solutions than CW2 algorithm on average in this warehouse layout. The biggest difference between GA and CW2 results is 6%. This is achieved in two cases, both of them when number of orders is 10 and maximum number of items per order is 5.

Both algorithms perform good on conditions when there are big number of items to pick and relatively small picker's capacity, and performed worse on conditions when picker has relatively big capacity and number of items which need to be picked is small.

The main difference from small warehouse, this layout has is number of rows. Small warehouse is set to have 10 rows of storage locations, while medium sized warehouse has 20 rows of storage locations. The number of rows doubled, while number of items is the same as for small warehouse. This leads to more diverse orders. This is before same orders now can be disperse in wider area, therefore, in medium sized warehouse picker has to travel more to pick same orders. This is the main reason, why overall results for both algorithms are worse than in small size warehouse. This is most visible in cases when number of items needed to pick is relatively small. As discussed before, when number of orders and items in them are small, and picker's capacity is relatively big, algorithms do not have much room to improve, because the most of improvements could be done just in queuing the items. When warehouse is larger, and same number of items is needed to pick, generated items are more scatter across the warehouse, making distance between items relatively similar. Therefore, different items picking queues result in similar tour distances. This is why CW2 and GA is not able to achieve as good results as in small warehouses.

**Table 3.8. Computation time needed in warehouse with 400 storage locations**

| Capacity | Max items per order | Orders | FCFS computation (ms) | CW2 computation (ms) | GA computation (ms) |
|---|---|---|---|---|---|
| 20 | 10 | 30 | 0 | 57 | 89674 |
| 10 | 5 | 30 | 0 | 38 | 41133 |
| 15 | 5 | 30 | 0 | 39 | 35512 |
| 20 | 10 | 20 | 0 | 17 | 29454 |
| 10 | 5 | 20 | 0 | 11 | 12195 |
| 15 | 5 | 20 | 0 | 11 | 12074 |
| 20 | 10 | 10 | 0 | 2 | 4180 |
| 10 | 5 | 10 | 0 | 1 | 1617 |
| 15 | 5 | 10 | 0 | 1 | 1574 |
| 20 | 10 | 5 | 0 | 1 | 606 |
| 10 | 5 | 5 | 0 | 0 | 325 |
| 15 | 5 | 5 | 0 | 0 | 317 |

Algorithms computation times are represented in Table 3.8. While CW2 algorithm computation times are relatively the same in all layouts, GA computation time is tended to increase with the layout size. In this case, GA take from 0.3s to 89s to complete its computations. 89 seconds is still good enough time to consider Genetic algorithm as valid solution for real world warehouses.

The performance of the GA when it took 89 seconds to complete is presented in Figure 3.3. The first best solution needs 3814 relative distance units to pick all of the items. GA is able to improve that solution until generation 44, when it gets solution which needs 3685 relative distance units to complete. After that algorithm is not able to improve for another 100 generations and terminates.

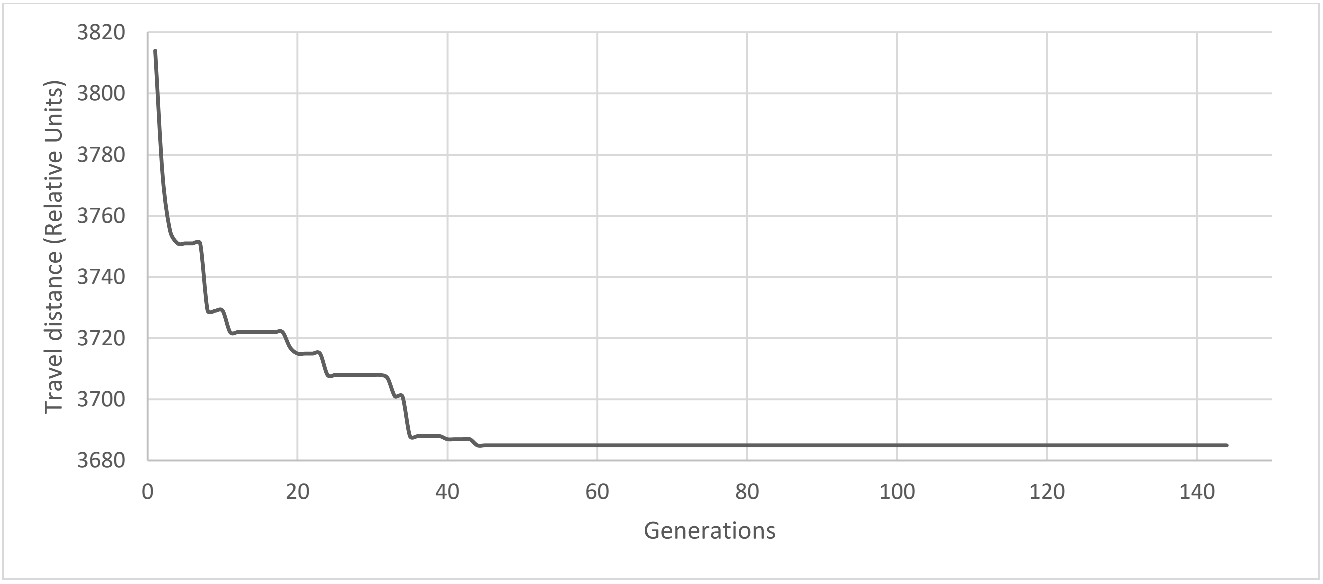


**Figure 3.3. GA performance in warehouse with 400 storage locations**

### 3.2.2 Warehouse with 600 Storage Locations

**Table 3.9. Algorithms performance in warehouse with 600 storage locations**

| Capacity | Max items per order | Orders | Savings CW2 | Savings GA | Difference (GA – CW2) |
|---|---|---|---|---|---|
| 20 | 10 | 30 | 12% | 12% | 0% |
| 20 | 10 | 20 | 9% | 9% | 0% |
| 20 | 10 | 10 | 7% | 7% | 0% |
| 20 | 10 | 5 | 4% | 5% | 1% |
| 15 | 5 | 30 | 36% | 36% | 1% |
| 15 | 5 | 20 | 30% | 30% | 0% |
| 15 | 5 | 10 | 20% | 22% | 2% |
| 15 | 5 | 5 | 23% | 23% | 0% |
| 10 | 5 | 30 | 36% | 37% | 1% |
| 10 | 5 | 20 | 30% | 30% | 0% |
| 10 | 5 | 10 | 20% | 22% | 2% |
| 10 | 5 | 5 | 23% | 23% | 0% |

On warehouse with 600 storage locations the performance difference by means of tour distance savings is smallest so far. As shown in Table 3.9, most of the times both algorithms create same results. Average savings difference in this case is just 0.6%.

CW2 generated solutions resulting in savings compared to FCFS from as low as 4% to as high as 36%. Average travel distance savings for CW2 algorithm is 20.6%.

Genetic algorithm was able to create solutions that ranges from 5% to 37%, averaging at 21.2% travel distance savings when compared to FCFS heuristic.

Both algorithms performed poorly on relatively big capacity conditions, however, both algorithms improved performance, when compared to previous warehouse layout. Both, CW2 and GA improved their average performance by 6.6% and 5.2% respectively. The key difference between current and previous layouts is the number of storage locations in one row. Previous layout has 20 storage locations and current layout has 30 storage locations. Increase in storage locations in one row also increases the length of one picking aisle. Therefore, it becomes more efficient to collect items in one picking aisle, than to jump from aisle to aisle like in previous layout, when there was no big difference, by means of travel distance if picker should stay in the same aisle, or go to another one. Therefore, randomly generated orders are less likely to create good solutions. So, both orders batching algorithms have more space to improve the solutions.

The maximum computation time of GA is increased once more. As shown in , computation time in worst case scenario is 97 seconds. Again, computation time for CW2 are more or less the same as before. 97 seconds is still not the bad time, however, CW2 generates very similar results 1650 times faster.

**Table 3.10. Computation times in warehouse with 600 storage locations**

| Capacity | Max items per order | Orders | FCFS computation (ms) | CW2 computation (ms) | GA computation (ms) |
|---|---|---|---|---|---|
| 20 | 10 | 30 | 0 | 58 | 96910 |
| 10 | 5 | 30 | 0 | 39 | 41406 |
| 15 | 5 | 30 | 0 | 39 | 34359 |
| 20 | 10 | 20 | 0 | 19 | 25807 |
| 15 | 5 | 20 | 0 | 11 | 12274 |
| 10 | 5 | 20 | 0 | 11 | 10949 |
| 20 | 10 | 10 | 0 | 3 | 4247 |
| 10 | 5 | 10 | 0 | 1 | 1681 |
| 15 | 5 | 10 | 0 | 1 | 1633 |
| 20 | 10 | 5 | 0 | 0 | 624 |
| 10 | 5 | 5 | 0 | 1 | 344 |
| 15 | 5 | 5 | 0 | 0 | 313 |

The longest computation is represented in Figure 3.4. It shows that GA made great improvements, however it basically reached its solution in generation 37. After that few minor improvements were found in generations 49 and 80. After that GA was not able to improve the solution, therefore it terminates on generation 180.

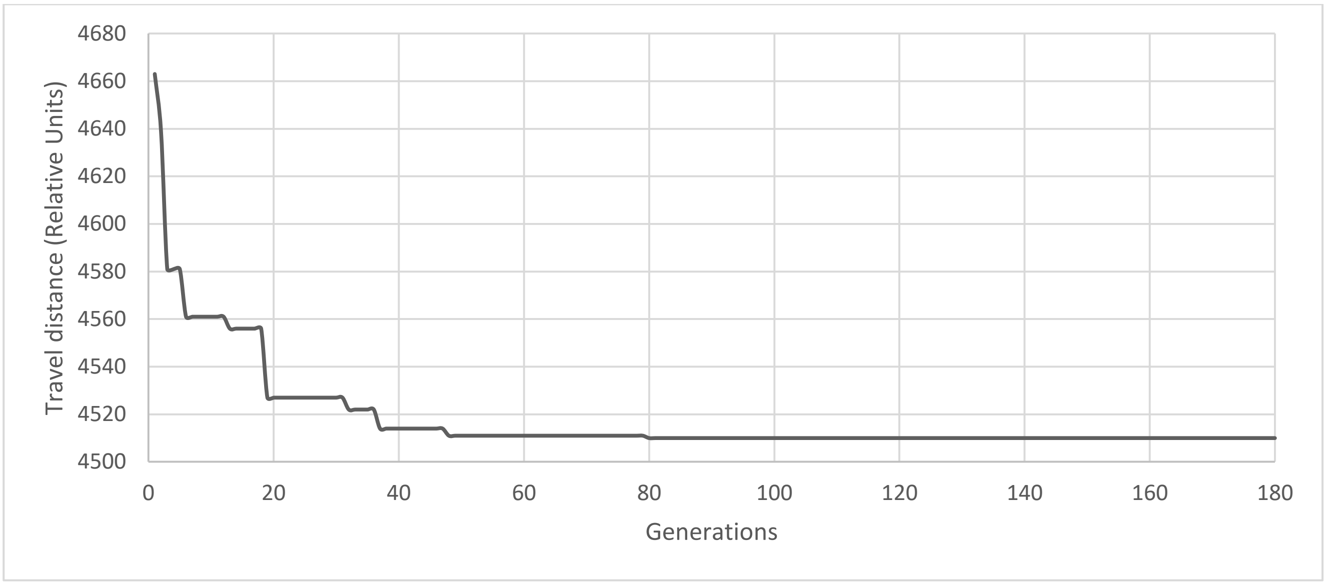


**Figure 3.4. GA performance in warehouse with 600 storage locations**

### 3.2.3 Warehouse with 800 Storage Locations

The results for warehouse with 800 storage locations are presented in Table 3.11. In this case GA is still able to generate same or better results than CW2 algorithm. The difference between saving varies from 0% to 6%. GA generates 1.5% shorter routes than CW2 on average in this case.

**Table 3.11. Results for the warehouse with 800 storage locations**

| Capacity | Max items per order | Orders | Savings CW2 | Savings GA | Difference (GA – CW2) |
|---|---|---|---|---|---|
| 20 | 10 | 30 | 13% | 14% | 1% |
| 20 | 10 | 20 | 11% | 11% | 0% |
| 20 | 10 | 10 | 9% | 9% | 0% |
| 20 | 10 | 5 | 8% | 8% | 0% |
| 15 | 5 | 30 | 45% | 45% | 0% |
| 15 | 5 | 20 | 34% | 36% | 2% |
| 15 | 5 | 10 | 14% | 20% | 6% |
| 15 | 5 | 5 | 17% | 17% | 0% |
| 10 | 5 | 30 | 45% | 45% | 0% |
| 10 | 5 | 20 | 34% | 36% | 2% |
| 10 | 5 | 10 | 14% | 20% | 6% |
| 10 | 5 | 5 | 17% | 17% | 0% |

Both algorithms show good performance in this layout. Best savings for both is 45% compared when compared to FCFS algorithm. CW2 and GA are able to generate almost two times shorter routes than FCFS. Like before both algorithms performs best when there is large amount of orders and small number of items, the picker can carry. Also, worst performance is achieved when capacity is big and number of items is small.

As mentioned earlier CW2 is able to save 45% travel distance at most and 8% at least cases. On this warehouse layout CW2 generates 22% shorter distance than FCFS on average.

Genetic algorithm results also range from 8% to 45% shorter distances than FCFS. GA on average generates 23.5% better results.

Computation time needed for GA is slightly increased when compared to time needed for warehouse with 600 storage locations. Computation times are represented in Table 3.12. GA needs from 0.3s to 98s to complete. Mostly, computation time depends on number of orders and maximum number of items per order. In large warehouse where small percentage savings could mean hundreds of meters or even kilometers, 98 seconds is reasonable amount of time. Therefore, GA is still valid solution for real time warehouses.

**Table 3.12. Computation times of warehouse with 800 storage locations**

| Capacity | Max items per order | Orders | FCFS computation (ms) | CW2 computation (ms) | GA computation (ms) |
|---|---|---|---|---|---|
| 20 | 10 | 30 | 0 | 58 | 97863 |
| 15 | 5 | 30 | 0 | 39 | 48320 |
| 10 | 5 | 30 | 0 | 39 | 35602 |
| 20 | 10 | 20 | 0 | 18 | 24312 |
| 15 | 5 | 20 | 0 | 11 | 13383 |
| 10 | 5 | 20 | 0 | 11 | 12189 |
| 20 | 10 | 10 | 0 | 3 | 4459 |
| 10 | 5 | 10 | 0 | 6 | 1793 |
| 15 | 5 | 10 | 0 | 1 | 1636 |
| 20 | 10 | 5 | 0 | 1 | 644 |
| 10 | 5 | 5 | 0 | 0 | 343 |
| 15 | 5 | 5 | 0 | 0 | 335 |

## 3.3 Large Warehouses

Large warehouses are represented by three layouts each having 900, 1200 and 1500 storage locations respectively. Each layout has 30 rows of local storages. The layouts differ in storage locations per row. Layouts have 30, 40 and 50 storage locations per row respectively.

### 3.3.1 Warehouse with 900 Storage Locations

**Table 3.13. Results in warehouse with 900 storage locations**

| Capacity | Max items per order | Orders | Savings CW2 | Savings GA | Difference (GA – CW2) |
|---|---|---|---|---|---|
| 20 | 10 | 30 | 13% | 13% | 0% |
| 20 | 10 | 20 | 5% | 8% | 3% |
| 20 | 10 | 10 | 2% | 7% | 5% |
| 20 | 10 | 5 | 9% | 9% | 0% |
| 15 | 5 | 30 | 27% | 30% | 3% |
| 15 | 5 | 20 | 24% | 24% | 0% |
| 15 | 5 | 10 | 19% | 19% | 0% |
| 15 | 5 | 5 | 28% | 28% | 0% |
| 10 | 5 | 30 | 27% | 30% | 3% |
| 10 | 5 | 20 | 24% | 24% | 0% |
| 10 | 5 | 10 | 19% | 19% | 0% |
| 10 | 5 | 5 | 28% | 28% | 0% |

Results for the warehouse with 900 storage locations are presenter in Table 3.13. Results show that Genetic algorithm generates same or better results than CW2 algorithm. GA produces up to 5% better orders picking tours by means of total travel distance than CW2. On average GA overcomes the CW2 by 1.2%. The biggest distance difference between GA and CW2 can be observed in cases, when small capacity is used. In all other cases algorithms produce almost identical results.

Savings of CW2 algorithm are distributed between 2% and 28%. This algorithm is able to create 19% better result on average when compared to FCFS algorithm.

Genetic algorithm creates results, which ranges from 7% to 30%. It is able to generate 20% better results on average when compared to FCFS.

**Table 3.14. Computation time needed in warehouse with 900 storage locations**

| Capacity | Max items per order | Orders | FCFS computation (ms) | CW2 computation (ms) | GA computation (ms) |
|---|---|---|---|---|---|
| 20 | 10 | 30 | 0 | 64 | 93778 |
| 15 | 5 | 30 | 0 | 42 | 56211 |
| 10 | 5 | 30 | 0 | 41 | 52206 |
| 20 | 10 | 20 | 0 | 17 | 29201 |
| 10 | 5 | 20 | 0 | 12 | 14298 |
| 15 | 5 | 20 | 0 | 11 | 13402 |
| 20 | 10 | 10 | 0 | 2 | 4974 |
| 10 | 5 | 10 | 0 | 3 | 1896 |
| 15 | 5 | 10 | 0 | 2 | 1866 |
| 20 | 10 | 5 | 0 | 0 | 742 |
| 15 | 5 | 5 | 0 | 0 | 384 |
| 10 | 5 | 5 | 0 | 0 | 372 |

As it could be noticed in , the computation time needed for GA to complete is almost the same as in warehouse with 800 storage locations. Computation times ranges from 0.4s to 94s. Computation time needed for CW2 is slightly bigger than on previous warehouse layout, however GA still needs 1360 times more time to complete its computations on average.

The performance of GA when it needed most of the time for computation is presented in Figure 3.5. As it is seen GA was able to found better solutions until generation 44. After that it was not able to improve the solution, therefore terminated after next 100 generations. Most of the time the algorithm was not able to find better solutions, therefore number of generations could be optimized for faster computation.

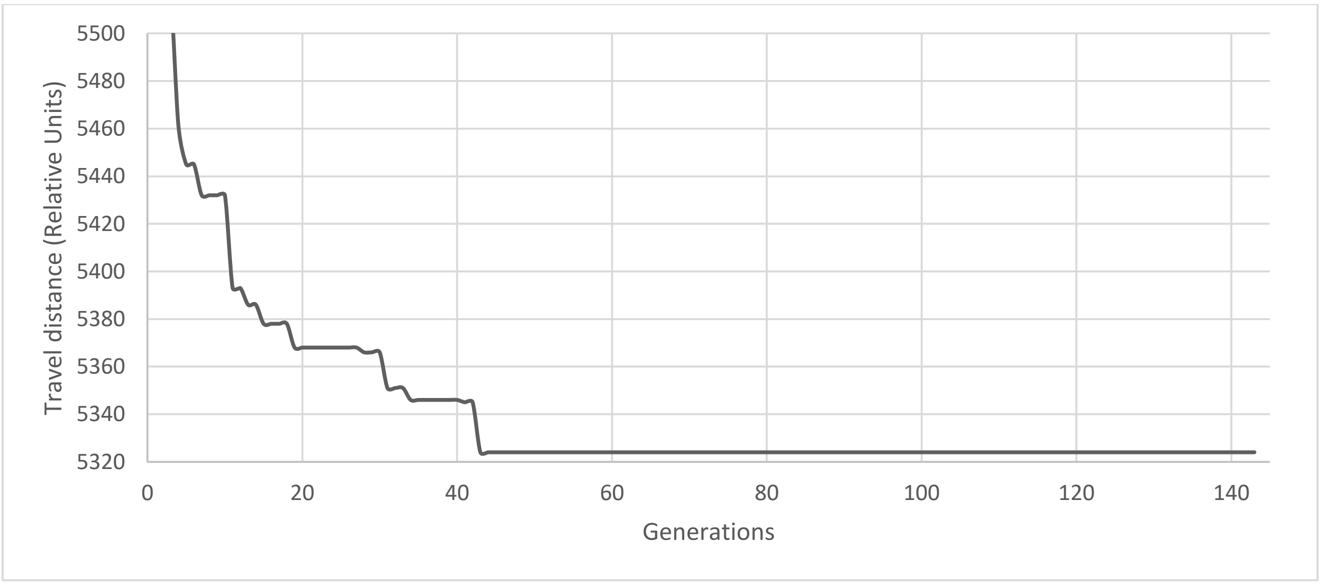


**Figure 3.5. GA performance in warehouse with 900 storage locations**

### 3.3.2 Warehouse with 1200 Storage Locations

**Table 3.15. Performance results in warehouse with 1200 storage locations**

| Capacity | Max items per order | Orders | Savings CW2 | Savings GA | Difference (GA – CW2) |
|---|---|---|---|---|---|
| 20 | 10 | 30 | 14% | 15% | 1% |
| 20 | 10 | 20 | 10% | 10% | 0% |
| 20 | 10 | 10 | 10% | 10% | 0% |
| 20 | 10 | 5 | 10% | 10% | 0% |
| 15 | 5 | 30 | 28% | 30% | 2% |
| 15 | 5 | 20 | 25% | 25% | 0% |
| 15 | 5 | 10 | 13% | 18% | 5% |
| 15 | 5 | 5 | 10% | 10% | 0% |
| 10 | 5 | 30 | 28% | 28% | 0% |
| 10 | 5 | 20 | 25% | 27% | 2% |
| 10 | 5 | 10 | 13% | 18% | 5% |
| 10 | 5 | 5 | 10% | 10% | 0% |

Warehouse containing 1200 storage locations is built using 30 locations rows and 40 storage locations in each row. The results for this layout are presented in Table 3.15. Once again, GA created same or better results than CW algorithm. The difference between results of two algorithms is 5% at most. The average GA improvement over CW2 is 1.3%.

The CW2 algorithm created results which are from 10% to 28% better than FCFS. On average CW2 is 16% better than FCFS on this particular layout.

Genetic algorithm was able to improve solutions from 10% to 30% and 18% on average when compared to FCFS.

**Table 3.16. Computation times in warehouse with 1200 storage locations**

| Capacity | Max items per order | Orders | FCFS computation (ms) | CW2 computation (ms) | GA computation (ms) |
|---|---|---|---|---|---|
| 20 | 10 | 30 | 0 | 65 | 103161 |
| 10 | 5 | 30 | 0 | 45 | 60991 |
| 15 | 5 | 30 | 0 | 42 | 49409 |
| 20 | 10 | 20 | 0 | 20 | 30418 |
| 10 | 5 | 20 | 0 | 12 | 17625 |
| 15 | 5 | 20 | 0 | 13 | 15644 |
| 20 | 10 | 10 | 0 | 3 | 5532 |
| 10 | 5 | 10 | 0 | 1 | 2063 |
| 15 | 5 | 10 | 0 | 1 | 2011 |
| 20 | 10 | 5 | 0 | 0 | 675 |
| 15 | 5 | 5 | 0 | 0 | 367 |
| 10 | 5 | 5 | 0 | 0 | 354 |

As it is shown in , the computation time of GA ranges from 0.4s to 103s. Again, the increase in warehouse size also increases the time needed for calculations. The more time algorithm needs to compute its solution, the less applicable it is in real world situations. 103 seconds is pretty long time for algorithm to complete. However, considering possible improvements on travel distance and also initial constraint of 5min as maximum allowed time, it is still valid to use this algorithm in real warehouse situations.

### 3.3.3 Warehouse with 1500 Storage Locations

The last large warehouse layout containing 1500 storage locations is built using 30 rows and 50 storage locations per row, making this layout the largest one in this experiment. The results of algorithms performance in such warehouse is represented in Table 3.17.

In this layout, Genetic algorithm was able to create same or by up to 8% better results when compared to CW2. On average GA was better than CW2 by 3%.

CW2 algorithm was able to produce results ranging from 7% to 29%. On average this algorithm is able to produce 16% better results than FCFS algorithm. The Genetic algorithm's results are from 7% to 33% and 19% on average better than FCFS.

Same as in previous examples, both algorithms performed best in cases when picker's capacity is relatively small and number of items that are needed to pick is relatively big. The worst performance was observed in cases when capacity is big and number of items is relatively small.

Comparing all large warehouse layouts, the best performance was observed in largest layout. This happens due to longer picking aisles, which makes harder for random orders generator to create good solutions at random.

**Table 3.17. Performance of the algorithms in warehouse with 1500 storage locations**

| Capacity | Max items per order | Orders | Savings CW2 | Savings GA | Difference (GA – CW2) |
|---|---|---|---|---|---|
| 20 | 10 | 30 | 15% | 17% | 2% |
| 20 | 10 | 20 | 14% | 14% | 0% |
| 20 | 10 | 10 | 11% | 11% | 0% |
| 20 | 10 | 5 | 7% | 7% | 0% |
| 15 | 5 | 30 | 29% | 32% | 3% |
| 15 | 5 | 20 | 13% | 18% | 5% |
| 15 | 5 | 10 | 13% | 18% | 5% |
| 15 | 5 | 5 | 15% | 15% | 0% |
| 10 | 5 | 30 | 29% | 33% | 4% |
| 10 | 5 | 20 | 13% | 21% | 8% |
| 10 | 5 | 10 | 13% | 18% | 5% |
| 10 | 5 | 5 | 15% | 15% | 0% |

As presented in , computation times in this case are a bit smaller than on previous layout, however do not differ significantly. In this case GA needed up to 101 seconds to complete its computation. CW2 algorithm in this layout was 1350 times faster than GA. However, 101 seconds is good time, when considered that GA generates 3% better results on average in this particular layout.

Time used for computation could lead to big savings in picker's travel distance, resulting in more efficient orders picking.

**Table 3.18. Computation times in warehouse with 1500 storage locations**

| Capacity | Max items per order | Orders | FCFS computation (ms) | CW2 computation (ms) | GA computation (ms) |
|---|---|---|---|---|---|
| 20 | 10 | 30 | 0 | 65 | 101096 |
| 10 | 5 | 30 | 0 | 44 | 55525 |
| 15 | 5 | 30 | 0 | 42 | 41569 |
| 20 | 10 | 20 | 0 | 19 | 33567 |
| 10 | 5 | 20 | 0 | 12 | 19612 |
| 15 | 5 | 20 | 0 | 12 | 13032 |
| 20 | 10 | 10 | 0 | 2 | 5234 |
| 15 | 5 | 10 | 0 | 2 | 1916 |
| 10 | 5 | 10 | 0 | 2 | 1906 |
| 20 | 10 | 5 | 0 | 1 | 694 |
| 15 | 5 | 5 | 0 | 0 | 486 |
| 10 | 5 | 5 | 0 | 0 | 440 |

The longest computation of the GA is represented in Figure 3.6. In this case GA was able to found near best solution in 12 generations, latter improvements were not so drastic. GA reached its optimal solution in generation 44 and terminated in generation 144. Therefore, again, most of the time, GA was not able to improve current best solution.

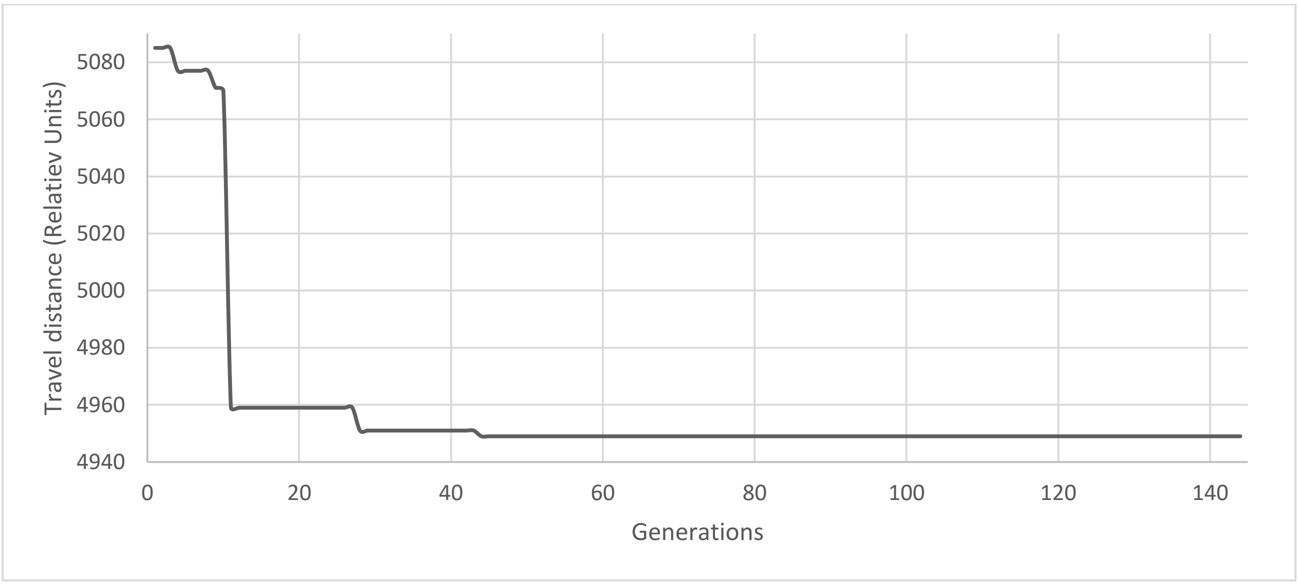


**Figure 3.6. GA performance in warehouse with 1500 storage locations**

## 3.4 Insights on the Results

The results showed that suggested Genetic algorithm is able to create same or better results when compared to the CW2 algorithm on same conditions. Though the GA is much slower than CW2 algorithm, the time needed for the calculations is still reasonable, therefore making the GA good option for the adoption in real warehouse. Genetic algorithm was able to improve FCFS results from 5% to 45%. On average GA was able to overcome the FCFS by 20%. The biggest difference in tour distance the Genetic algorithm was able to generate when comparing to CW2 algorithm was 11%. On average GA was able to overcome the CW2 algorithm by 2%.

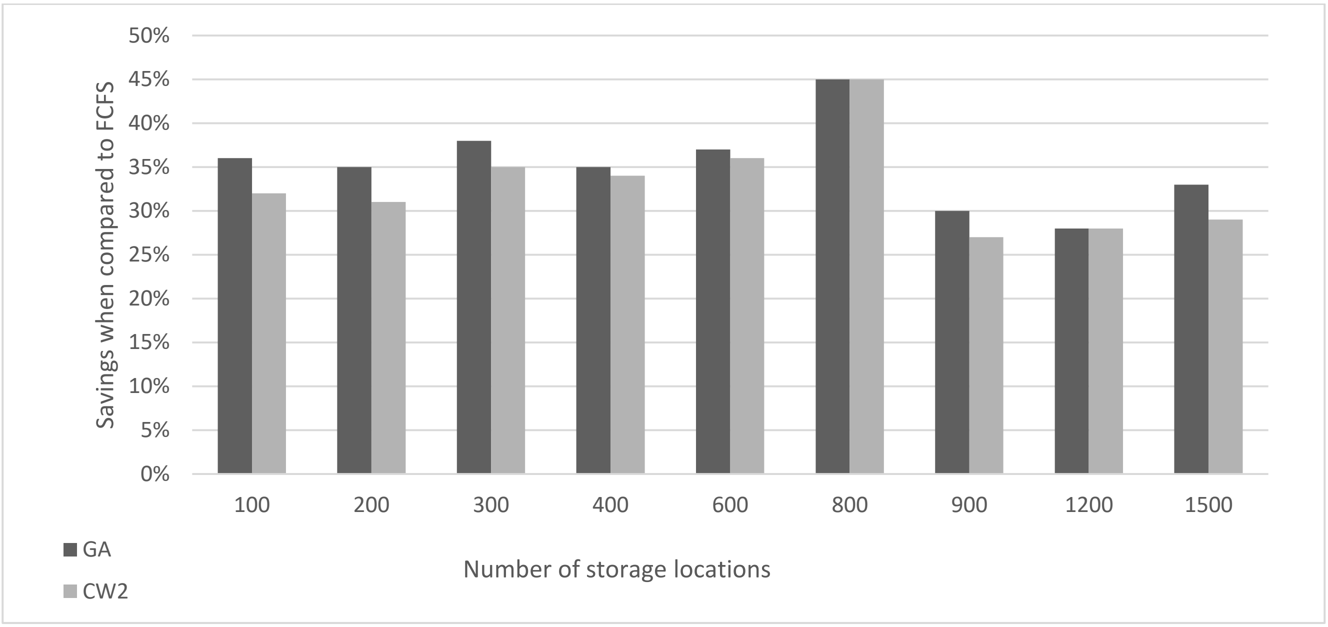


**Figure 3.7. Algorithms performance on best conditions**

The best and the worst parameters for the Genetic algorithm was identified. The GA performed best in cases, when there was a large number of items which need to be picked and relatively small number of items picker can carry. In these experiments, such parameters were: 30 orders, 5 items picker's capacity and 5 items at maximum in one order. The performance of both GA and CW2 algorithms on these parameters on different warehouses are represented in Figure 3.7.

On these parameter, the average performance of CW2 is 33% and average performance of GA is 35% shorter distances when compared to FCFS respectively. This makes these two algorithm very powerful tool to increase the efficiency of the orders picking operation in warehouses, which works on such or similar conditions.

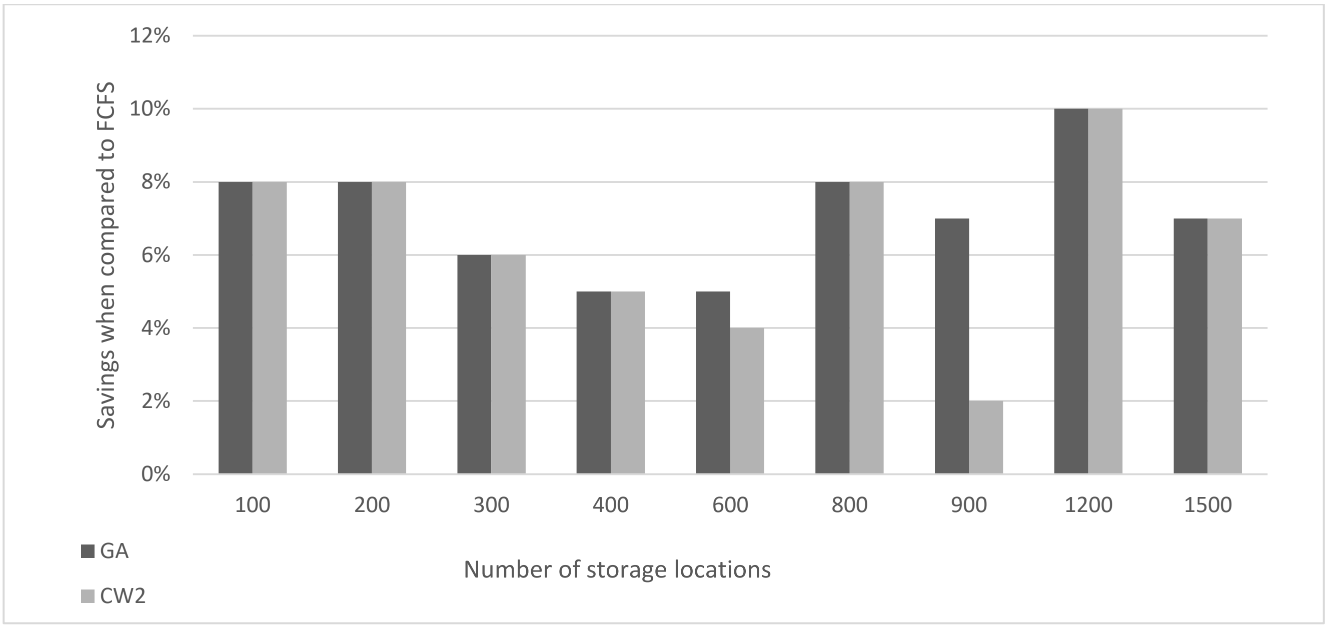


**Figure 3.8. Algorithms performance on worst conditions**

The worst parameters are identified to be opposite to the best parameters. The main attributes of bad parameters for such algorithms are the ones, with large picker's capacity and small number of items which need to picked. The worst vary from layout to layout, but all have: 20 items picker's capacity and 5 or 10 orders. On these parameters, the performance of GA varies from 5% to 10%, while CW2 performance ranges from 2% to 10%. The average saved distance on such conditions for both GA and CW2 algorithms are 7% and 6% respectively.

Another factor which has big influence on how good algorithms perform is the number of storage locations in one row. The results show that the more storage locations are in one row, the better is the performance of the algorithms. As mentioned before, when the number of storage locations in one row is small, the distances between randomly generated items are more similar and it is less dependent on the picking aisles. When number of storage locations per row increases, random generated items are closer to each other when they are in the same picking aisle. Therefore, FCFS can generate good results randomly on layouts with low number of storage locations on one row, though the other algorithms have small room for improvement. The opposite is for layouts with long picking aisles. The longer the aisles, the smaller is the probability that FCFS will generate good solution on random. Therefore, other algorithms have much more room to perform and improve the solution of FCFS.

As mentioned before, another big factor in determining how good the algorithm is, is the computation time. Results indicated that conditions that needs the most of the time to compute are:  30 orders, and 20 items picker's capacity. In this case, computation time ranges from 68s to 103s. The comparison of computation times on different warehouse layouts between best conditions parameters and longest computation time parameters is represented in Figure 3.9. On average, best conditions computation takes 50% of time needed for the longest computation on the same warehouse layout.

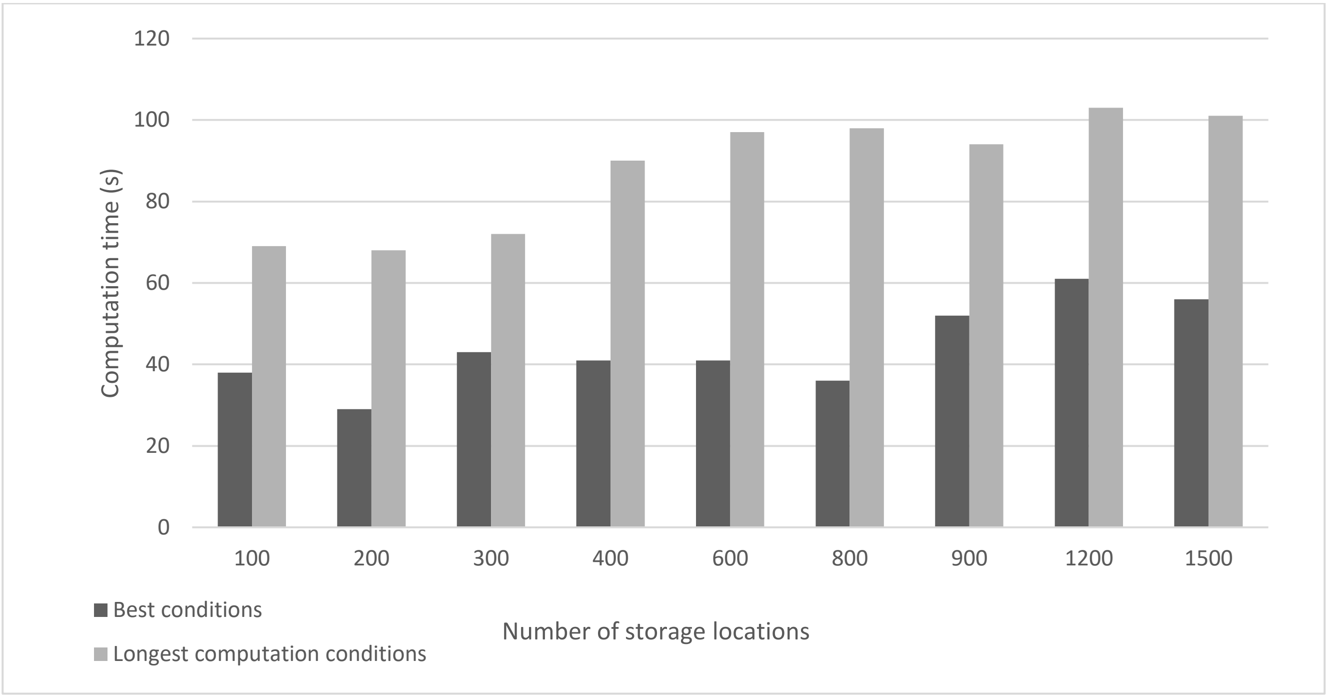


**Figure 3.9. Comparison of computation times between best and longest computation conditions**

As the results suggest, the computation time depend on the number of possible combinations of items picking queue. The larger the capacity and more items there are, the more possible combinations there could be. Also, the results show that another factor which has a big influence on computation time is the total number of storage locations in the warehouse.

## 3.5 Enhancements of Genetic Algorithm

The analysis of GA performance suggests some points where the algorithm could still be improved. As the most GA performance graphs showed, algorithm finds its optimum solution in pretty early stages. For example, in warehouse with 1500 storage locations, GA was able to found near best solution in just 12 generations. It also was able to improve that until generation 45, however these improvements are minor.

There are two key points that need to be considered. On one hand, the convergence to optimum solution on early generations could indicate that the number of generation is set to high, therefore, the most of the computation time and resources are wasted. There are very few situations, when GA needed 200 or more generations. On most of the time it was able to find the best solution in 60 generations. Therefore, the reduction of number of generations could be good option to make the algorithm faster.

On the other hand, early optimum solution could also indicate, that with each later generation, algorithm is moving away from better solutions, due to its random nature. In this case, one bad cross-over or mutation, can create population that is far from current best solution, therefore it is harder for the algorithm to find new better solutions. There could be many solutions for such problem, however in this only case two possible options are considered.

In the first one, better performance could be achieved by injection of best solution into the population with each generation. In this case, the worst solution in the population would be replaced by current best solution. This would still let the algorithm to improve by itself, however with a bit more guidance.

The second solution, which could lead to better solutions is a bit more proactive. In this case, all new individuals in the populations would be created from the current best solution and one other parent. In this case, the Genetic algorithm is more similar to the local search algorithms than to normal random evolution based algorithms.

To verify these options, additional round of experiments is conducted. All 9 different layouts are tested. The parameters for these experiments selected to be: 20 orders, 5 items capacity and 5 maximum items per one order.

The results of these experiments are presented in Table 3.19 and Table 3.20. The first one shows the distance comparison of the original GA and two updated GAs differs. Data is presented as how much percent the performance of the updated algorithm is better when compared to original GA.

As results suggest, there was no big improvement when using best solution injection technique. Only on the smallest layout the new algorithm was able to create better result than the original GA. On all other cases the results were the same.

The results of best solution cross-over technique show the same again, results are very similar to results that are generated by original GA. The biggest difference could be seen on 300 storage locations layout where updated algorithm produced 4% worse distance comparing to original GA.

**Table 3.19. Travel distance savings of updated GAs compared to the original**

| | **100** | **200** | **300** | **400** | **600** | **800** | **900** | **1200** | **1500** |
|---|---|---|---|---|---|---|---|---|---|
| Best solution injection | 1% | 0% | 0% | 0% | 0% | 0% | 0% | 0% | 0% |
| Best solution cross-over | -1% | 0% | -4% | 0% | 0% | -1% | 0% | 0% | 0% |

How much generations saved when computing the results for both updated algorithms in all layouts when compared to original algorithm are presented in Table 3.20. As the results suggest, the best solution injection technique was able to reduce the number of generation needed most of the times. When compared to the original GA, best solution injection modification was able to save 8% of generations on average. The generations range from 25% shorter calculations to 9% longer calculations when compared to original algorithm.

The generations result for the GA with best solution cross-over modification resulted in smaller or same number generations needed to find the solution. The generations savings ranges from 0% to 45%. On average best solution cross-over was able to save 23% of computation resources. Therefore, this modification will be applied to the final algorithm, which will be tested in real warehouse environment simulation against the algorithm which is used in this warehouse.

**Table 3.20. Generations savings on updated GAs compared to the original**

| | **100** | **200** | **300** | **400** | **600** | **800** | **900** | **1200** | **1500** |
|---|---|---|---|---|---|---|---|---|---|
| Best solution injection | 16% | -5% | -8% | 25% | 15% | -9% | 12% | 28% | -2% |
| Best solution cross-over | 34% | 3% | 3% | 29% | 36% | 21% | 37% | 45% | 0% |

Another improvement that will be applied on final version of the algorithm is the use of the multicore architecture, which allows to utilize all available computing power on the machine. The original algorithm run on one of processors core, the multicore architecture allows to use all processor cores. The experiments showed, that when using computer with 4 physical cores, the computation times are reduced 3 to 4 times.

# 4. Real Warehouse Simulation

To know, if developed Genetic algorithm could be applicable in real world warehouse, it is determined to create a real warehouse simulation. This simulation mimics the processes of real warehouse. It covers the layout of the warehouse, storage locations of specific items, distances, orders structure and other parameters that are specific to this warehouse. The order picking algorithm that is used in this warehouse is also simulated. Therefore, it makes possible to compare the Genetic algorithm, developed in this work, to the algorithm which is used in real world warehouse, in conditions that are close to ones that could be found in real warehouse.

## 4.1 Parameters of the Warehouse Simulation

The selected warehouse is located in Kaunas district. The main production that is stored in it is the various components for the electrical circuits: circuit breakers, relays, contactors, mounting cabinets, etc.  The layout of the warehouse is presented in Figure 4.1. Layout of the warehouse

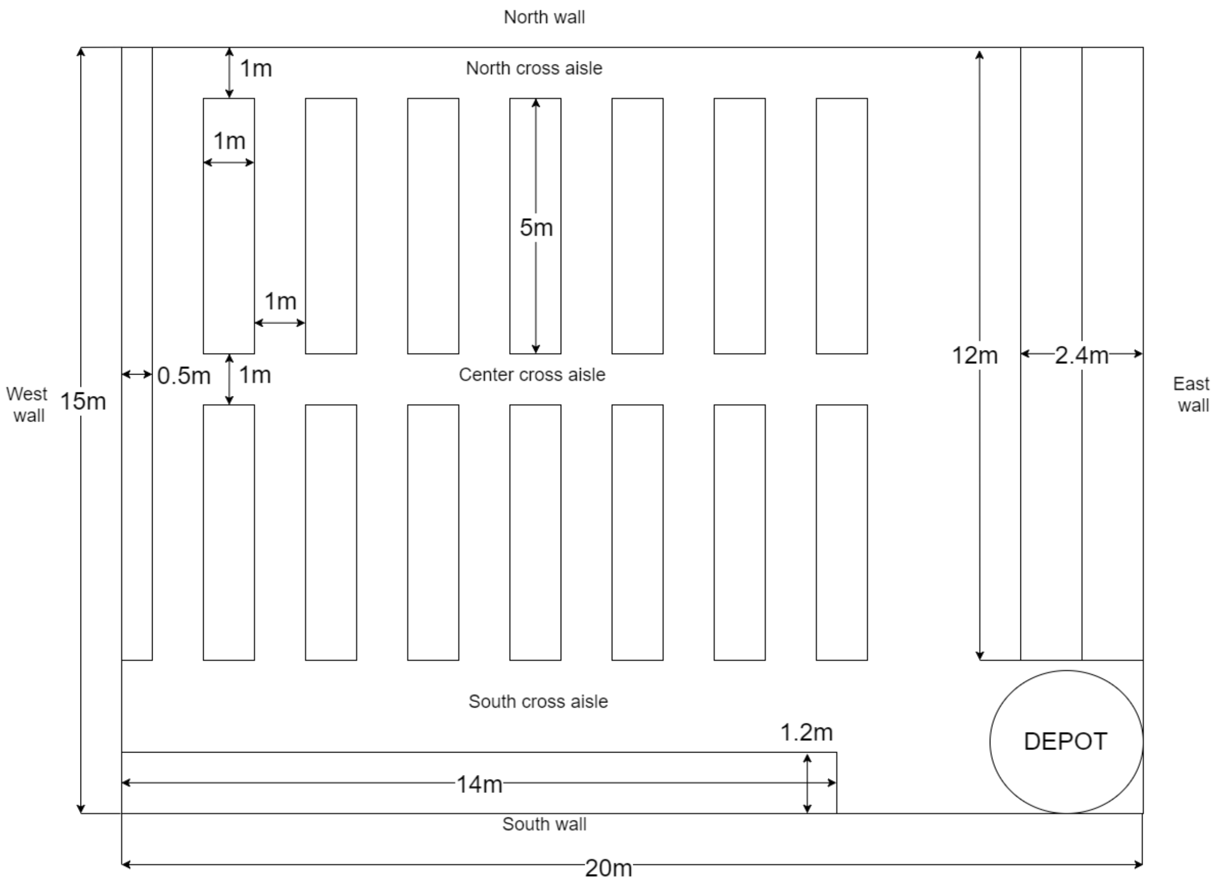


**Figure 4.1. Layout of the warehouse**

The area of selected warehouse is 300m$^2$. It has eight picking aisles and three cross aisles. Each picking aisle is 1m wide. North and center cross aisles are 1m wide and south cross aisle is 1.8m wide. Items in the middle of the warehouse are stored on 4 stories racks. The dimensions of those racks are 5x1 meters. There also one line of the racks by the western wall. The dimensions of those racks are 12x0.5 meters. The items which are stored on pallets on the floor are lined up by the southern wall and in two lines by the eastern wall of the building. The depot and the entrance are on the south-east corner of the warehouse. For easier reference, the racks are numbered as shown in Figure 4.2.

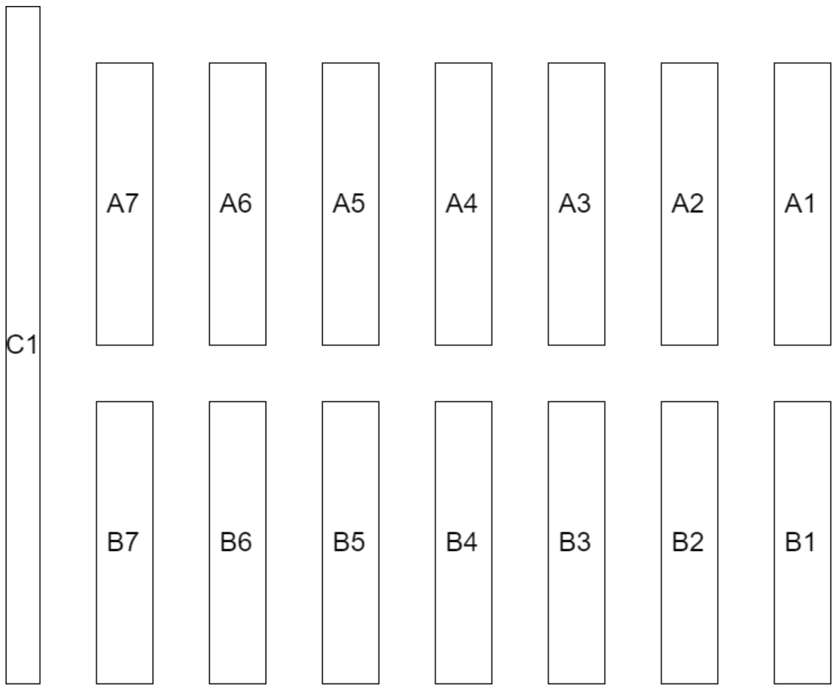


**Figure 4.2. Racks numbering**

The items on the racks are stored in the way that the most frequently ordered items are nearer to the depot. The most ordered items are stored in: A1, A2, A3, B1, B2, B3 and B4. These items are ordered 50% of the times. Items that are ordered less frequently (35%) are stored on racks: A4, A5, A6, B5, B6 and B7. The items that are more specific and are ordered least frequent are stored in A7 and C1.

For the sake of simplicity, it is assumed that one item takes 1m x 0.5m area to store. Different items that are store at same position but different elevation on the rack, will be considered as same item. Therefore, the real warehouse has 320 storage locations, but the simplified one, which will be used for calculations has only 80.

The sizes of items could also vary; hence it is also assumed, that picker can carry 10 items that are stored on the racks, and only one item that is stored on the floor. Therefore, the size of one item which is stored on the racks is 1, the size of item stored on the floor is 10 and the capacity of the picker is 10.

The usual orders picking workflow in this warehouse is as follows: first orders are buffered. This process usually takes one day. On the next day buffered orders are batched and picked. If order has a high priority, such order is not buffered, but picked immediately on the same day without batching. The number of orders received in one day may vary from 20 to 80. To cover all cases, there is four experiments conducted, each having different number of orders: 20, 40, 60 and 80.

The orders batching algorithm used in this warehouse could be described as optimized FCFS algorithm. First all orders are grouped in to the batched, based on their arrival time and pickers capacity same as in simple FCFS algorithm. After that, batches, which still have free space, by means of picker's capacity, are batched together if the sum of their orders is not violating the capacity constraint. Therefore, this algorithm utilizes the picker's capacity very well.

The main measurement of the performance in this experiment is the length of the tour, when all orders in the buffer are picked. In this case, differently than in previous chapter, meters will be used as the measurement units. The other performance index is the computation time. If the difference of the tour lengths between two algorithms is similar, computation time will determine if developed GA algorithm is better or worse than current picking algorithm used in this warehouse. To do so, it is determined that average walking speed of picker is 1.4m/s. Therefore, the minimum difference between tour lengths, to specify GA as the better algorithm can be calculated as:

$$\Delta L_{min} = 1.4 \times (T_{GA} - T_{FCFS})$$

Where:

- $\Delta L_{min}$ is minimum length difference to specify Genetic algorithm as better.
- $T_{GA}$ is Genetic algorithm computation time
- $T_{FCFS}$ is warehouse batching algorithm computation time

## 4.2 Results

After concluding four experiments with different number of orders, the results were obtained. It is important to note, that computation time depends on the machine the algorithm is running on. In this particular case, experiments where conducted on laptop running Ubuntu 17.04 OS, having 8GB of RAM and Intel i5 processor having four physical cores. The results of experiment when 20 orders are used is shown in Table 4.1. In this experiment 20 orders buffer was used. As it is shown in **Klaida! Nerastas nuorodos šaltinis.** The modified FCFS algorithm, which is currently used in the warehouse batched orders in the way, that picker would need to travel 661m to pick all of them. The computation time needed for this algorithm is really small, therefore it is considered to be 0 seconds.

**Table 4.1. Results of experiment with 20 orders**

| | Distance (m) | Computation time (s) | Difference (m) | Minimum Difference (m) | Saved (%) |
|---|---|---|---|---|---|
| FCFS | 661 | 0 | 88 | 10 | 13% |
| GA | 573 | 7 | | | |

The Genetic algorithm was able to batch same orders in the way, picker would need to go 573m to pick all items in these orders. Therefore, GA produced 13% shorter tour distance. The algorithm took 7 seconds to run. The minimum required difference is calculated to be 10m. The actual difference is 88m, making the GA the better solution in this particular case.

Next the experiment with 40 orders was conducted. The results of this experiment are shown in Table 4.2. This time FCFS was able to produce the solution, which would need 1580m to pick all items, while GA generated the solution which would need 1353m to pick all items. The Genetic algorithm generated 14% shorter than FCFS. It also needed 32 seconds to complete. This indicates that minimum saved distance GA needs is 45m. While actual saved distance is 227m, this again makes GA better solution than current used algorithm in this particular case.

**Table 4.2. Results of experiment with 40 orders**

| | Distance (m) | Computation time (s) | Difference (m) | Minimum Difference (m) | Saved (%) |
|---|---|---|---|---|---|
| FCFS | 1580 | 0 | 227 | 45 | 14% |
| GA | 1353 | 32 | | | |

Next, experiment with 60 orders was done. The results of this experiment are shown in Table 4.3. When orders buffer is 60, current warehouse batching algorithm was able to generate solution, which needs 2387m to pick all items. The GA was able to create solution, which tour length is 1877m. Genetic algorithm needed 121 seconds to finish. This makes minimum required distance 170m. Actual difference is 510m, therefore 21% travel distance was saved by GA when comparing to current algorithm used in this warehouse. Again, GA proved to be better batching algorithm in this situation.

**Table 4.3. Results of experiment with 60 orders**

| | Distance (m) | Computation time (s) | Difference (m) | Minimum Difference (m) | Saved (%) |
|---|---|---|---|---|---|
| FCFS | 2387 | 0 | 510 | 170 | 21% |
| GA | 1877 | 121 | | | |

Results of last experiment, when orders buffer is set to 80, are represented in Table 4.4. The modified FCFS algorithm produced the solution, which travel tour is 3259m. The Genetic algorithm was able to create solution, which tour length is 2674m. The GA needed 343 seconds to complete its calculations, therefore the minimum required distance savings is 480m. Actual distance savings are 585m or 18% of the solution generated by FCFS.  This once again makes GA better solution for this case.

**Table 4.4. Results of experiment with 80 orders**

| | Distance (m) | Computation time (s) | Difference (m) | Minimum Difference (m) | Saved (%) |
|---|---|---|---|---|---|
| FCFS | 3259 | 0 | 585 | 480 | 18% |
| GA | 2674 | 343 | | | |

All four experiments show the same, in this work developed Genetic algorithm is able to produce better solutions than currently in this warehouse used algorithm. The tour length produced by Genetic algorithm is 17% shorter on average. While computation time of GA increases with the number of orders, the savings this algorithm produces overcomes this problem. In all cases, actual savings where bigger than minimum required distance difference. The computation time depends on the computer running the algorithm directly. These results may be different on slower machine. In such case, the computation times could be so long, that in time the algorithm finishes its computation, picker could be finished most of the picking using current algorithm used in warehouse, which do not require much computing power.

# Conclusions

1. The analysis of literature indicated that the most often used solutions for order batching problem, both in numerical experiments and in real warehouses are the constructive heuristics and metaheuristics. Two algorithms were selected to serve as benchmarks. First is the First Come First Serve (FCFS) algorithm. This algorithm is used in many warehouses due to its simplicity and speed. However, this algorithm is random at its nature, therefore can't produce really good solutions constantly. FCFS is used as the benchmark for lowest acceptable performance. The other algorithm is the second adaptation of Clarke and Wright algorithm (CW2) for orders batching problem. This algorithm is used in many numerical experiments as well as in real warehouses. CW2 algorithm is proven to be one of the best performing algorithms for orders batching problem, therefore it represents the desired performance.  The developed algorithm should perform as well as CW2 algorithm or even better.

2. The new algorithm for orders batching problem is developed. It is population based algorithm called Genetic algorithm (GA). GA uses cross-over and mutation processes to modify its current population and create a new, better one. After specified number of generation GA stops and best solution is determined.

3. 108 different experiments where conducted to compare the performance of selected benchmark algorithms and developed Genetic algorithm. The results showed that developed GA is able to create same or better results than CW2 algorithm on same conditions. However, the GA is much slower than CW2 algorithm, the time needed for the calculations is still reasonable, therefore making the GA good option for the adoption in real warehouse. Genetic algorithm was able to improve FCFS results from 5% to 45%. On average GA was able to overcome the FCFS by 20%. The biggest difference in tour distance the Genetic algorithm was able to generate when comparing to CW2 algorithm was 11%. On average GA was able to overcome the CW2 algorithm by 2%.

4. The best and the worst parameters for the Genetic algorithm was identified. The GA performed best in cases, when there was a large number of items which need to be picked and relatively small number of items picker can carry. In these experiments, such parameters were: 30 orders, 5 items picker's capacity and 5 items at maximum in one order. On these parameters, the average performance of CW2 is 33% and average performance of GA is 35% shorter distances when compared to FCFS respectively. The worst parameters are identified to be opposite to the best parameters. The main attributes of bad parameters for such algorithms are the ones, with large picker's capacity and small number of items which need to picked.

The worst vary from layout to layout, but all have: 20 items picker's capacity and 5 or 10 orders. On these parameters, the performance of GA varies from 5% to 10%, while CW2 performance ranges from 2% to 10%. The average saved distance on such conditions for both GA and CW2 algorithms are 7% and 6% respectively. Results of the experiments also showed the possible enhancements for GA. The GA was modified to use best solution cross-over method, to improve its computation time. For the same reason, GA was updated to use multicore processors architecture. These improvements led to up to 7 times faster computation times.

5. Simulated testing environment based on real warehouse was implemented. It was also determined that the algorithm used in this warehouse is optimized FCFS algorithm. The GA and optimized FCFS was than tested and their results were compared. Four experiments where conducted, each representing different demand. In all experiments, Genetic algorithm generated shorter travel tour distances. The long computation times of Genetic algorithm became neglectable because of its savings. The average saved distance, when comparing GA to the optimized FCFS algorithm, used in the warehouse, was 17%. Therefore, developed Genetic algorithm is the better solution than optimized FCFS, which is used in this particular warehouse.

# References

[1] S. H. Storbjerg, T. D. Brunoe and K. Nielsen, "Engineering Change Management and Transition," Department of Mechanical and Manufacturing Engineering, Aalborg University, Aalborg, 2017.

[2] M. Murray, "Order Picking In The Warehouse," 03 2017. [Online]. Available: https://www.thebalance.com/order-picking-in-the-warehouse-2221190.

[3] J. Tompkins, J. White, J. Bozer and Y. Frazzele, Facilities Planning 3rd. Edition, New Jersey: Wiley, 2003.

[4] T. L. Chen, C. Y. Cheng, Y. Y. Chen and L. K. Chan, "An efficient hybrid algorithm for integrated order batching, sequencing and routing problem," *International Journal of Production Economics,* pp. 158-167, 2015.

[5] R. de Koster, T. Le-Duc and K. Roodbergen, "Design and Control of Warehouse Order Picking: A Literature Review," *European Journal of Operational Research,* vol. 182, no. 2, pp. 481-501, 2007.

[6] C. Huber, "Throughput Analysis of Manual Order Picking Systems with Congestion Consideration," Scientific Pulishing, 2011.

[7] D. Battini, M. Calzavara, A. Persona and F. Sgarbossa, "Additional effort estimation due to ergonomic conditions in order picking systems," *International Journal of Production Research,* 2016.

[8] M. Napolitano, "2012 Warehouse/DC Operations Survey: Mixed signals," *Modern Materials Handling,* no. 11, pp. 48-56, 2012.

[9] R. Elbert, T. Franzke, C. Glock and E. Grosse, "The effects of human behavior on the efficiency of routing policies in order picking: the case of route deviations," *Computers & Industrial Engineering,* 2016.

[10] C. Wheeler, "The 8 Best Order Picking Methods," 16 6 2014. [Online]. Available: http://www.newcastlesys.com/blog/bid/348476/order-picking-methods-and-the-simplest-ways-to-minimize-walking-infographic.

[11] S. Henn, S. Koch, K. Doerner, C. Strauss and G. Wascher, "Metaheuristics for the order batching problem in manual order picking systems," *Business Research,* vol. 3, no. 1, pp. 82-105, 2010.

[12] G. Wascher, "Order Picking: A Survey of Planning Problems and Methods," Springer, Berlin, 2004.

[13] N. Gademann and S. van de Velde, "Order Batchin to Minimize Total Travel Time in Parrallel-Aisle Warehouse," *IIE Transactions,* vol. 33, no. 5, pp. 385-398, 2005.

[14] J. Ericson, "NP-Hard Problems," in *Algorithms*, 2014, p. 3.

[15] "Constructive Heuristic," Wikipedia, January 2017. [Online]. Available: https://en.wikipedia.org/wiki/Constructive_heuristic. [Accessed 28 March 2017].

[16] R. Ruben and F. Jacobs, "Batch Construction Heuristics and Storage Asignment Strategies for Walk/Ride and Pick Systems," *Management Science,* vol. 45, no. 4, pp. 575-596, 1999.

[17] J. Pan and S. Liu, "A Comparative Study of Order Batching Algorihms," *Omega - International Journal of Management Science,* vol. 23, no. 6, pp. 691-700, 1995.

[18] E. Elsayed, "Algorithms for Optimal Material Handling in Automatic Warehousing Systems," *International Journal of Production Research,* vol. 19, no. 5, pp. 525-535, 1981.

[19] Y. Ho and Y. Tseng, "A Study orn Order-Batching Methods for Order-Picking in a Distribution Center with two Cross Aisles," *International Journal of Production Research,* vol. 44, no. 17, pp. 3391-3417, 2006.

[20] E. Elsayed and O. Unal, "Order Batching Algorithms and Travel-Time Estimation for Automated Storage/Retrieval Systems," *International Journal of Production Research,* vol. 27, no. 7, pp. 1097-1114, 1989.

[21] S. Luke, "Esentials of Metaheuristics," October 2015. [Online]. Available: https://cs.gmu.edu/~sean/book/metaheuristics/Essentials.pdf. [Accessed 28 March 2017].

[22] M. Albareda-Sambola, A. Alonso-Ayuso, E. Molina and C. de Blas, "Varible Neighborhood Search for Order Batching in a Warehouse," *Asia-Pacific Journal of Operational Research,* vol. 26, no. 5, pp. 655-683, 2009.

[23] S. Henn and G. Wäsher, *Tabu Search Heuristics for the rder Batching Problem in Manual Order Picking Systems,* Magdeburg, 2010.

[24] M. C. Hsu, Y. Chen and M. Chen, "Batching Orders in Warehouses by Minimizing Travel Distance with Genetic Algorithms," *Computers in Industry,* vol. 56, no. 2, pp. 169-178, 2005.

[25] I. van Nieuwenhuyse and R. de Koster, "Evaluating Order Throughut Time in 2-block Warehouses with Time Window Batching," *International Journal of Product Economics,* vol. 46, no. 22, pp. 654-664, 2009.

[26] D. Gibson and G. Shap, "Order Batching Procedures," *European Journal of Operational Research,* vol. 58, no. 1, pp. 57-67, 1992.

[27] M. Rosenwein, "A Comparison of Heuristics for the Problem of Batching Orders for Warehouse Selection," *International Journal of Production Research,* vol. 34, no. 3, pp. 657-664, 1996.

[28] TriFactor, "Order Picking Methods: Order Fulfillment - Order Picking Systems," TriFactor, 2008. [Online]. Available: http://www.trifactor.com/Material-Handling-Engineering-and-System-Design/Order-Fulfillment-Order-Picking-System-Design/Order-Picking-Methods.

[29] Y. Bozer and J. Kile, "Order Batching and Walk-and-Pick Order Picking Systems," *International Journal of Production Research*, vol. 46, no. 7, pp. 1887-1909, 2008.

# Appendices

**Results of algorithms in warehouse with 100 storage locations**

| Capacity | Max items per order | Orders | FCFS distance (Relative Units) | CW2 distance (Relative Units) | GA distance (Relative Units) |
|---|---|---|---|---|---|
| 20 | 10 | 30 | 1806 | 1541 | 1541 |
| 20 | 10 | 20 | 1243 | 1106 | 1106 |
| 20 | 10 | 10 | 591 | 544 | 544 |
| 20 | 10 | 5 | 234 | 202 | 202 |
| 15 | 5 | 30 | 775 | 530 | 513 |
| 15 | 5 | 20 | 471 | 361 | 334 |
| 15 | 5 | 10 | 230 | 193 | 187 |
| 15 | 5 | 5 | 144 | 116 | 116 |
| 10 | 5 | 30 | 775 | 530 | 495 |
| 10 | 5 | 20 | 471 | 361 | 334 |
| 10 | 5 | 10 | 230 | 193 | 190 |
| 10 | 5 | 5 | 144 | 116 | 116 |

**Results of algorithms in warehouse with 200 storage locations**

| Capacity | Max items per order | Orders | FCFS distance (Relative Units) | CW2 distance (Relative Units) | GA distance (Relative Units) |
|---|---|---|---|---|---|
| 20 | 10 | 30 | 2983 | 2587 | 2573 |
| 20 | 10 | 20 | 2085 | 1879 | 1879 |
| 20 | 10 | 10 | 1032 | 954 | 954 |
| 20 | 10 | 5 | 372 | 330 | 330 |
| 15 | 5 | 30 | 1226 | 850 | 818 |
| 15 | 5 | 20 | 734 | 550 | 522 |
| 15 | 5 | 10 | 349 | 337 | 304 |
| 15 | 5 | 5 | 245 | 205 | 205 |
| 10 | 5 | 30 | 1226 | 850 | 803 |
| 10 | 5 | 20 | 734 | 550 | 522 |
| 10 | 5 | 10 | 349 | 337 | 299 |
| 10 | 5 | 5 | 245 | 205 | 205 |

**Results of algorithms in warehouse with 300 storage locations**

| Capacity | Max items per order | Orders | FCFS distance (Relative Units) | CW2 distance (Relative Units) | GA distance (Relative Units) |
|---|---|---|---|---|---|
| 20 | 10 | 30 | 3653 | 3169 | 3163 |
| 20 | 10 | 20 | 2510 | 2302 | 2302 |
| 20 | 10 | 10 | 1252 | 1177 | 1177 |
| 20 | 10 | 5 | 477 | 425 | 425 |
| 15 | 5 | 30 | 1549 | 1006 | 956 |
| 15 | 5 | 20 | 970 | 643 | 598 |
| 15 | 5 | 10 | 462 | 363 | 350 |
| 15 | 5 | 5 | 279 | 196 | 196 |
| 10 | 5 | 30 | 1549 | 1006 | 929 |
| 10 | 5 | 20 | 970 | 643 | 598 |
| 10 | 5 | 10 | 462 | 363 | 350 |
| 10 | 5 | 5 | 279 | 196 | 196 |

**Results of algorithms in warehouse with 400 storage locations**

| Capacity | Max items per order | Orders | FCFS distance (Relative Units) | CW2 distance (Relative Units) | GA distance (Relative Units) |
|---|---|---|---|---|---|
| 20 | 10 | 30 | 4210 | 3685 | 3685 |
| 20 | 10 | 20 | 2976 | 2677 | 2677 |
| 20 | 10 | 10 | 1506 | 1403 | 1403 |
| 20 | 10 | 5 | 560 | 534 | 534 |
| 15 | 5 | 30 | 1843 | 1211 | 1213 |
| 15 | 5 | 20 | 1066 | 840 | 818 |
| 15 | 5 | 10 | 473 | 454 | 427 |
| 15 | 5 | 5 | 322 | 292 | 292 |
| 10 | 5 | 30 | 1843 | 1211 | 1191 |
| 10 | 5 | 20 | 1066 | 840 | 823 |
| 10 | 5 | 10 | 473 | 454 | 427 |
| 10 | 5 | 5 | 322 | 292 | 292 |

**Results of algorithms in warehouse with 600 storage locations**

| Capacity | Max items per order | Orders | FCFS distance (Relative Units) | CW2 distance (Relative Units) | GA distance (Relative Units) |
|---|---|---|---|---|---|
| 20 | 10 | 30 | 5116 | 4524 | 4510 |
| 20 | 10 | 20 | 3521 | 3202 | 3202 |
| 20 | 10 | 10 | 1725 | 1598 | 1598 |
| 20 | 10 | 5 | 641 | 615 | 606 |
| 15 | 5 | 30 | 2210 | 1422 | 1408 |
| 15 | 5 | 20 | 1322 | 931 | 935 |
| 15 | 5 | 10 | 616 | 494 | 482 |
| 15 | 5 | 5 | 372 | 288 | 288 |
| 10 | 5 | 30 | 2210 | 1422 | 1393 |
| 10 | 5 | 20 | 1322 | 931 | 935 |
| 10 | 5 | 10 | 616 | 494 | 482 |
| 10 | 5 | 5 | 372 | 288 | 288 |

**Results of algorithms in warehouse with 800 storage locations**

| Capacity | Max items per order | Orders | FCFS distance (Relative Units) | CW2 distance (Relative Units) | GA distance (Relative Units) |
|---|---|---|---|---|---|
| 20 | 10 | 30 | 5300 | 4593 | 4548 |
| 20 | 10 | 20 | 3716 | 3304 | 3303 |
| 20 | 10 | 10 | 1921 | 1746 | 1746 |
| 20 | 10 | 5 | 733 | 677 | 677 |
| 15 | 5 | 30 | 2437 | 1342 | 1333 |
| 15 | 5 | 20 | 1519 | 1002 | 966 |
| 15 | 5 | 10 | 698 | 599 | 559 |
| 15 | 5 | 5 | 384 | 318 | 318 |
| 10 | 5 | 30 | 2437 | 1342 | 1334 |
| 10 | 5 | 20 | 1519 | 1002 | 965 |
| 10 | 5 | 10 | 698 | 599 | 559 |
| 10 | 5 | 5 | 384 | 318 | 318 |

**Results of algorithms in warehouse with 900 storage locations**

| Capacity | Max items per order | Orders | FCFS distance (Relative Units) | CW2 distance (Relative Units) | GA distance (Relative Units) |
|---|---|---|---|---|---|
| 20 | 10 | 30 | 6094 | 5324 | 5324 |
| 20 | 10 | 20 | 4142 | 3905 | 3800 |
| 20 | 10 | 10 | 2047 | 2000 | 1905 |
| 20 | 10 | 5 | 855 | 780 | 780 |
| 15 | 5 | 30 | 2639 | 1916 | 1854 |
| 15 | 5 | 20 | 1632 | 1239 | 1233 |
| 15 | 5 | 10 | 858 | 699 | 698 |
| 15 | 5 | 5 | 476 | 344 | 344 |
| 10 | 5 | 30 | 2639 | 1916 | 1853 |
| 10 | 5 | 20 | 1632 | 1239 | 1233 |
| 10 | 5 | 10 | 858 | 699 | 698 |
| 10 | 5 | 5 | 476 | 344 | 344 |

**Results of algorithms in warehouse with 1200 storage locations**

| Capacity | Max items per order | Orders | FCFS distance (Relative Units) | CW2 distance (Relative Units) | GA distance (Relative Units) |
|---|---|---|---|---|---|
| 20 | 10 | 30 | 6288 | 5420 | 5379 |
| 20 | 10 | 20 | 4322 | 3899 | 3899 |
| 20 | 10 | 10 | 2162 | 1954 | 1954 |
| 20 | 10 | 5 | 831 | 745 | 745 |
| 15 | 5 | 30 | 2632 | 1893 | 1844 |
| 15 | 5 | 20 | 1685 | 1260 | 1262 |
| 15 | 5 | 10 | 912 | 789 | 745 |
| 15 | 5 | 5 | 482 | 434 | 434 |
| 10 | 5 | 30 | 2632 | 1893 | 1901 |
| 10 | 5 | 20 | 1685 | 1260 | 1236 |
| 10 | 5 | 10 | 912 | 789 | 745 |
| 10 | 5 | 5 | 482 | 434 | 434 |

**Results of algorithms in warehouse with 1500 storage locations**

| Capacity | Max items per order | Orders | FCFS distance (Relative Units) | CW2 distance (Relative Units) | GA distance (Relative Units) |
|---|---|---|---|---|---|
| 20 | 10 | 30 | 5931 | 5033 | 4949 |
| 20 | 10 | 20 | 4122 | 3539 | 3539 |
| 20 | 10 | 10 | 2079 | 1857 | 1857 |
| 20 | 10 | 5 | 767 | 711 | 711 |
| 15 | 5 | 30 | 2384 | 1683 | 1611 |
| 15 | 5 | 20 | 1383 | 1208 | 1138 |
| 15 | 5 | 10 | 726 | 630 | 593 |
| 15 | 5 | 5 | 428 | 362 | 362 |
| 10 | 5 | 30 | 2384 | 1683 | 1607 |
| 10 | 5 | 20 | 1383 | 1208 | 1095 |
| 10 | 5 | 10 | 726 | 630 | 593 |
| 10 | 5 | 5 | 428 | 362 | 362 |