Bachelor Project

Czech Technical University in Prague

F3

Faculty of Electrical Engineering
Department of Computer Science

Automated warehouse expedition scheduling

Jan Kalina

Supervisor: Ing. Martin Schaefer
May 2019

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

I. OSOBNÍ A STUDIJNÍ ÚDAJE

Příjmení: **Kalina** Jméno: **Jan** Osobní číslo: **460511**

Fakulta/ústav: **Fakulta elektrotechnická**

Zadávající katedra/ústav: **Katedra počítačů**

Studijní program: **Otevřená informatika**

Studijní obor: **Software**

II. ÚDAJE K BAKALÁŘSKÉ PRÁCI

Název bakalářské práce:

Rozvrhování vyskladňování z automatizovaného skladu

Název bakalářské práce anglicky:

Automated warehouse expedition scheduling

Pokyny pro vypracování:

Consider a scenario of a given automated warehouse with a given set of products to dispatch. The problem to consider is to schedule a daily expedition plan with respect to given operation constraints of the warehouse. The properties to consider are: priority to production handling, limited throughput, robustness to failures of individual robots.
1) Research existing methods for relevant scheduling problems
2) Discuss and propose optimization methods for the formalized expedition scheduling problem.
3) Implement the proposed methods in a provided simulation tool and evaluate the performance of the methods in simulation.

Seznam doporučené literatury:

[1] Pinedo, Michael L. Scheduling: theory, algorithms, and systems. Springer, 2016.
[2] Brucker, Peter. Scheduling algorithms. Vol. 3. Berlin: Springer, 2007.
[3] Gu, Jinxiang, Marc Goetschalckx, and Leon F. McGinnis. &quot;Research on warehouse design and performance evaluation: A comprehensive review.&quot; European Journal of Operational Research 203.3 (2010): 539-549.

Jméno a pracoviště vedoucí(ho) bakalářské práce:

Ing. Martin Schaefer, centrum umělé inteligence FEL

Jméno a pracoviště druhé(ho) vedoucí(ho) nebo konzultanta(ky) bakalářské práce:

Datum zadání bakalářské práce: **23.01.2019** Termín odevzdání bakalářské práce: **24.05.2019**

Platnost zadání bakalářské práce: **20.09.2020**

Ing. Martin Schaefer podpis vedoucí(ho) práce	podpis vedoucí(ho) ústavu/katedry	prof. Ing. Pavel Ripka, CSc. podpis děkana(ky)

III. PŘEVZETÍ ZADÁNÍ

Student bere na vědomí, že je povinen vypracovat bakalářskou práci samostatně, bez cizí pomoci, s výjimkou poskytnutých konzultací.
Seznam použité literatury, jiných pramenů a jmen konzultantů je třeba uvést v bakalářské práci.

.	
Datum převzetí zadání	Podpis studenta

Acknowledgements

I want to thank Ing. Martin Schaefer for patiently supervising my thesis and constantly putting me on the right tracks. I would also like to thank Andrea Švancarová for keeping me company and making even stressful times fun. And finally, thanks to my family, who always motivates me and supports me in everything I do.

Declaration

Abstract

This thesis deals with the optimization problem of daily expedition scheduling in an automated warehouse with a focus on robustness and satisfaction of given requirements. The problem consists of a truck allocation to expedition ramps and scheduling of unloading of predetermined products. We introduce related scheduling problems and the methods to solve them. Our method for schedule evaluation is then presented together with the proposed heuristic construction method in combination with local search techniques. Finally, the proposed method is implemented in the provided simulation tool, and constructed schedule is evaluated based on results from the simulation. Results from the simulation show that the proposed method generates a schedule that satisfies given requirements and is relatively robust to delays.

Keywords: scheduling, automated warehouse, optimization, simulation

Supervisor: Ing. Martin Schaefer

Abstrakt

Tato práce se zabýva optimalizací problému rozvrhování denní expedice v automatizovaném skladu s důrazem na robustnost a vyhovění daným požadavkům. Problém zahrnuje přiřazení kamionů na rampy a naplánovaní vyskladnění předem daných produktů. Představujeme relevatní problémy rozvrhování a způsoby jejich řešení. Metoda pro evaluaci rozvrhu je poté prezentována společně s heuristic construction metodou v kombinaci s lokálním prohledáváním. Kombinace těchto metod je navržena k tvorbě rozvrhu expedicí. Navržená metoda je implementována v dodaném simulátoru a vzniklý rozvrh je evaluován na základě výsledků ze simulace.

Klíčová slova: rozvrhování, automatizovaný sklad, optimaliza, simulace

Překlad názvu: Rozvrhování vyskladňování z automatizovaného skladu

Contents

Figures

Tables

Chapter 1

Introduction

Scheduling is a well studied and practical topic. It is widely used in manufacturing facilities, warehouses, which are discussed in this thesis, and many other industries where proper scheduling can play a great role in their success. Despite its importance and years of research, scheduling is quite a challenging problem, especially when it comes to real-world problems where finding an optimal schedule is usually nearly impossible due to its high computational complexity.

Since the beginning of research in scheduling, many methods and approaches were developed for finding schedules close to optimal schedule in a reasonable time, but what is a good or optimal schedule? Determining the objective of a schedule is a problem on its own. The usual objective of scheduling is to minimize the makespan, which is the duration of a schedule. This objective alone is usually not sufficient because it does not consider the stochastic nature of real-world, where random events can change the schedule for the worse. In that case, the most suitable schedule might be the one that is more robust to random events, and makespan is not optimal, but still sufficient. Another reason why makespan or other simple objectives that are often mentioned in literature might not be sufficient is that some facilities may demand complex customized requirements tailored just for them. For example, in case of an automated warehouse, it may be crucial to not only finish expedition in required time but to also expedite items in the required order.

In this thesis, general methods for solving scheduling problems relevant to scheduling a daily expedition of the given automated warehouse configuration are discussed. Then, method and an objective of a schedule for the given automated warehouse and scenario is proposed. Finally, the proposed method is implemented and evaluated in the simulation.

1.1 The automated warehouse

There are many different types of automated warehouses with very different requirements, structures, and functionality. The problem of scheduling an expedition may depend on the structure and the type of an automated warehouse. The basic structure and functionality of an automated warehouse based on a real-world problem were provided for this thesis and can be described as follows.

The automated warehouse is part of a factory. The factory stores products to the automated warehouse from which they are at some point expedited.

Firstly, we describe production together with a layout of the automated warehouse and secondly, we describe the expedition process.

1.1.1 Production

The factory is running 24 hours a day and produces up to thousands of items throughout a whole day. The exact amount of produced items is unknown since not all items pass quality control. The time of item arrival at the warehouse is also unknown because items can be delayed for unknown reasons.

Items come in hundreds of different types, which may differ in size, material, or in other properties, which are not important for this thesis. Since the warehouse is automated, items must have suitable weight and size to be handled by a stacker crane and to be stored in high-bay racks, like the ones shown in Figure 1.1. Examples of such items are tires or pallets of different sodas. Produced items are directly put one-by-one on the conveyor leading to the warehouse, where an item is pushed off the conveyor to the production buffer of one of the aisles.

The automated warehouse has a given number of aisles. Each aisle has high-bay racks on both sides and is connected to two conveyors at the beginning of it. The first conveyor is moving items from an aisle to a conveyor junction and then one of the expedition ramps, and the second one is bringing in items from production. Each aisle has a single stacker crane operating on it.

Stacker cranes operate automatically and can either store an item from production if it is available or execute scheduled unloadings.

If stacker crane is requested to unload an item, it moves to the position of an item, grabs it and moves to the beginning of an aisle, where it puts the item on the conveyor leading to an expedition ramp. In case of an item arriving from production to an aisle's production buffer, stacker crane should retrieve this item from the production buffer and store in a free position in the aisle. If any production buffer overflows, all the production would have to be stopped, which can lead to serious financial losses. For that reason, stacker cranes prioritize storing an item over unloading. It means that as soon as a stacker crane finishes an operation in progress, it stores an item from production (if it is available at the buffer) even if it means delaying scheduled unloading.

1.1.2 Expedition

The expedition is spanned over a part of one day; we will further refer to this part as working hours. We are given information about when the working hours start and end. During the expedition, a truck can arrive at an expedition ramp at a scheduled time. Only one truck can be loaded at a time at each expedition ramp. There is also reserved time after which the next truck can arrive. Every truck requests a certain number of items of different types and order of these types in which they should be loaded into the truck for practical reasons. Which specific items will be expedited is known in advance and is expected that these items are distributed uniformly across all aisles.

Before each item reaches an expedition ramp, it arrives at a conveyor junction and then proceeds to the expedition ramp's scanner, where it needs to be scanned. A scanner is located at the end of the conveyor before each expedition ramp.

The order in which items are processed by a scanner corresponds to order at which they are loaded into a truck. Scanners accept an item periodically. Since the scanner is processing items in the given interval, there has to be space between items on conveyor based on the speed of the conveyor and the duration of the interval. If too many items arrive too soon after each other, they pile up and may cause the conveyor to stop, which should never happen. To prevent stopping of a conveyor, there is a small buffer. A scanner also accepts an item as soon as possible. The number of items that pile up in the buffer during a daily expedition is one of the measurements of robustness to delays.

We also simplify our problem and assume that workers at the loading ramp can always pick up an item coming from the scanner. That means that scanners then create bottlenecks for each expedition ramp where throughput is limited by the frequency at which scanner can process items.

On top of all these specifications, it is expected that the expedition should not last longer than the working hours. Additionally, the duration of a truck expedition should not take too much time; there is a constraint on how long the truck expedition should be. Parameters specifying these requests and the importance of these requests is something that a client should specify, and it has to be considered during the construction of a schedule.

Warehouse parameters

Several basic parameters were provided for the given automated warehouse. These parameters include horizontal and vertical speeds and accelerations of stacker cranes, dimensions of a warehouse, speed and length of conveyors, duration of scanning an item, duration of grabbing an item from a conveyor, duration of putting an item on a conveyor and exact location of each item stored in the warehouse. For the expedition, we know count, types, and order at which items should be loaded for each truck.

1.2 Goals and motivation

The goal of this thesis is to implement a scheduler that can produce a schedule in a reasonable time by assigning trucks planned for a day to expedition ramps, determining their order and the order at which items will be expedited

Figure 1.1: Example of an automated warehouse with stacker cranes. Licensed under CC-BY-SA-4.0 [1]

and finally determine the specific time at which stacker cranes should start unloading each item so that items arrive in the desired order to the expedition ramps. Since there is no or uncertain information about item production, we cannot incorporate production handling in our schedule, but knowing that production handling can delay unloading of an item, we can try to minimize the effect that production handling has on the schedule. To properly evaluate the schedule, it is integrated into simulation so that the created schedule can be observed in settings with production and evaluated based on it.

With a combination of the schedule and the simulation tool, we are also able to get data about the capabilities of the warehouse itself, which is valuable information when designing an automated warehouse [6]. Integration of the scheduler to the simulation tool also enriches the simulation tool and can be used in future projects.

1.3 Notation and terminology

Job A job is a term used in scheduling. Jobs are usually assigned execution time and machine that executes it. In this thesis, a job represents an item to be expedited or truck to be assigned to a ramp.
Machine A machine represents a stacker crane in this thesis.
Processing time (p_i) A time that a machine needs to process a job. In other words, a time needed to unload an item from its location to conveyor. This action consists of a trip from a conveyor to the item i, grabbing the item, trip back to the conveyor and putting the item on it. Values of p_i are calculated from speeds, accelerations, and dimensions of the warehouse. Processing time is never zero.
Scanning interval (s) The time needed for a scanner to process an item.
Traverse duration (t_ij) The time needed for an item i to get to a scanner

from this aisle by a conveyor.
Completion time (c_i) The time at which a scanner finishes scanning an item i.
Idle time (θ_i) After processing an item i at a scanner, the next item is allowed to be scheduled at that scanner after the idle time θ_i.
Assigned expedition ramp (r_i) A ramp to which an item i is scheduled on.
Position in expedition (pos_i) Position of an item i in the expedition order for its assigned expedition ramp.
Set of ramp indices (R)
Completion time of a truck (c_i^i) The time at which the last item demanded by a truck arrives at its expedition ramp.
Start of working hours ($start$) The beginning of working hours. From this time on, trucks can be scheduled to arrive.
End of working hours (end) The end of working hours. Ideally, every truck should be expedited by this time.
Stacker crane completion time (scc_i) The time at which stacker crane unloaded an item i.
Stacker crane start time (scs_i) The time at which stacker crane starts unloading an item from the warehouse.
Type identification of an item (Serial number) ($type_i$)
Expected types ($expedition_{i,j}$) An expected item type of an item leaving the arriving $i-th$ to a expedition ramp j.
Number of items to be expedited at an expedition ramp ($size_i$) Total count of requested items at an expedition ramp i.

Chapter 2

Problem statement

We split the problem of expedition scheduling in the automated warehouse into two different problems, trucks allocation, and item dispatching. The first problem of truck allocation is dealing only with the assignment of trucks to ramps. There are no requirements or constraints regarding order at which trucks should arrive at each ramp.

The second problem, item dispatching, occurs after the truck allocation, and it deals with determining which items should meet which requests from trucks and the assignment of completion times of item unloading. Since machines are predetermined, there is no need to select a machine which unloads an item. An item is already associated with a machine.

Both of these problems are described as machine models in the following sections, which are used in scheduling literature (See [12] or [4] for an overview of machine models). More formal formulations for both of these problems are included in the following sections.

2.1 Trucks allocation

This problem can be formulated as a parallel machine model.

There are r ramps in parallel and t trucks planned for a day. Ramps can be interpreted as machines in parallel and jobs as the whole expedition of each truck.

Since each ramp includes only one scanner, that acts as a bottleneck in the warehouse, the objective of this problem is to distribute expedition among ramps uniformly to utilize these ramps. In other words, workers at each expedition ramp should ideally work the same hours and do the same amount of work.

The problem can be formulated as assignment problem as follows:

$$\text{minimize} \quad \max(y_0, \ldots, y_r) \tag{2.1}$$

subject to

$$\sum_{i=0}^{n} x_{ij} = 1 \quad \text{for } j = 0, \ldots, r \tag{2.2}$$

$$\sum_{i=0}^{t} d_i \cdot x_{ij} = y_j \quad \text{for } j = 0, \ldots, r \tag{2.3}$$

Where x_{ij} is a binary decision variable, which indicates that the ith truck is scheduled on jth expedition ramp. By satisfying Equation 2.2 it is ensured that every truck is assigned to only one ramp. Equation 2.3 sets variable y_i to duration of expedition on the expedition ramp i. By minimizing Function 2.1 we minimize the makespan, which leads to good utilization of machines [12].

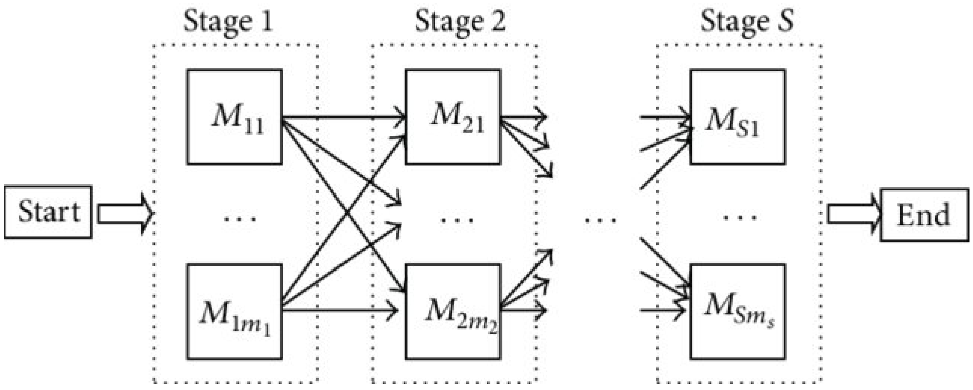

Figure 2.1: Flexible flow shop layout from article by Min Dai [5]. Licensed by CC BY 3.0.

2.2 Item dispatching

The problem of assigning completion times to jobs in the automated warehouse can be formulated as a special case of 2-stage flexible flow shop (FF2) with blocking or flexible flow line (FFL) with two stages as follows:

There are two work stations connected in series. The first work stations consist of m machines (stacker cranes) in parallel, and the second work station consists of r machines (scanners) also in parallel. Every job job_i needs to be processed on its predetermined machine, and processing of job_i takes p_i seconds. After job job_i is processed, it travels t_ij seconds to get to a scanner j where it needs to be processed too. Scanning time s is for every scanner the same. Jobs cannot wait between work stations. That alone is referred to as FFL. Figure 2.1 illustrates the flow of the FFL model.

Additionally, we need to ensure that an item i arrives at a truck in an order at which this item's type is requested and since this problem follows the truck allocation problem, orders at which items should arrive at expedition ramp are known. By assigning an order and a ramp to an item from a warehouse, we assign an item to a request. This leads to the following problem formulation.

For convenience, we formulate this problem as a constraint satisfaction problem as follows:

Variables:

$$V = \{c_0, c_1, \ldots, c_n, r_0, \ldots, r_m, pos_0, \ldots, pos_n\} \tag{2.4}$$

Domains:

$$
\begin{aligned}
&c_i \in C_i = \{start, ..., 86400\} && \text{for } i = 0, \ldots, n && (2.5)\\
&r_i \in R && \text{for } i = 0, \ldots, n && (2.6)\\
&pos_i \in \{1, \ldots, n\} && && (2.7)
\end{aligned}
$$

Constraints:

$$
\begin{aligned}
&pos_i < size_{r_i} && \text{for } i = 0, \ldots, n && \\
&pos_i < pos_j \land r_i = r_j \implies c_i < c_j && \text{for } i = 0, \ldots, n && (2.8)\\
&m_i = m_j \implies scc_i \leq scs_j \lor scc_j \leq scs_i && \text{for } i, j = 0, \ldots, n && (2.9)\\
&r_i = r_j \implies c_i + \theta_i \leq s_j \lor c_j + \theta_j \leq c_i - s && \text{for } i, j = 0, \ldots, n && (2.10)\\
&expedition_{pos_i, r_i} = type_i && \text{for } i = 0, \ldots, n && (2.11)\\
& && && (2.12)
\end{aligned}
$$

Where Constraint 2.9 ensures that a variable pos_i represent an order at which an item i is loaded at expedition ramp r_i. Constraints 2.10 and 2.11 secures that stacker cranes and scanners can process only one item at a time respectively. Finally, Constraint 2.12 checks if type of an item i is expected at expedition ramp r_i in pos_i-th order.

Main requirements of this problem are to make a schedule that fits into working hours of the warehouse and make the schedule robust to random events. For example, the arrival of an item from production can postpone scheduled job at a machine for a whole duration of storing the item, and that can lead to loading items in the wrong order and filling buffer at a scanner.

Chapter 3

Related work

Methods for solving various forms of scheduling problems were extensively studied for decades now. Many of these methods are summarized in Scheduling by M. Pinedo [12] and Scheduling Algorithm by P. Brucker[4]. We will mainly discuss methods from these books that can be applied to the flexible flow line or parallel machine model problems and some methods that are used in next chapters.

3.1 Overview of methods and approaches for finding optimal or nearly optimal schedule

Both problems we consider in this thesis are NP-hard [12]. To find the optimal schedule of FFL or parallel machine model, seemingly two general methods are used, Mixed Integer Programming (MIP) and Constraint Programming (CP) with a combination of pruning methods, an approximation of initial bounds and other general methods. MIP approach is often obsolete for real-world problems, where there are hundreds or thousands of jobs to schedule. CP is in many cases similar to MIP, but offers better flexibility for designing constraints and can be optimized for specific problems and because of it often outperforms MIP. CP is a very vast topic, and more about it can be found in Principles of Constraint Programming by K. Apt [3].

There are also some special cases of machine models and scheduling objectives for which we can construct an optimal schedule using different job dispatching rules or algorithms. Neither FFL scheduling nor parallel machine model of truck allocation problem can be optimally solved using some dispatching rules as far as we know, but we will take a look on some special cases that can give an idea what dispatching rules should perform well.

3.2 Minimizing makespan of parallel machine model

Since minimization of makespan of parallel machine model is NP-hard, it is often solved using CP or MIP approach. As we have shown in Section 2.1 if there are no special requirements, it can be easily formulated mathematically, which is required when using CP or MIP approach.

There are few construction heuristics or in other words dispatching rules, from which we can construct a schedule that guarantees an upper bound of the makespan. One of these heuristics is The Longest Processing Time first (LPT) dispatching rule. Using LPT rule, jobs are scheduled on any freed machine in decreasing order of their processing. According to R. L. Graham [11], LPT rule guarantees that:

$$\frac{C_{max}(LPT)}{C_{max}(OPTIMAL)} \leq \frac{4}{3} - \frac{1}{3m} \tag{3.1}$$

Where m is a number of parallel machines.

But like we said, to find an optimal schedule, we need to use some general methods like MIP or CP and LPT rule can be used for approximation of an upper bound.

3.3 Flexible flow line scheduling

To find an optimal schedule for FFL with even some basic objectives like makespan can be solved again by using some of the general methods like CP. In practice, it is often necessary to use some heuristics for choosing a job to dispatch. More about different heuristics for FFL and their application can be found in the article Heuristic Methods for Flexible Flow Line Scheduling from S. Kochnar [10].

A heuristic that is worth mentioning is Johnson's rule (also known as SPT(1)-LPT(2)) for minimizing the makespan of FFL with two stages and a single machine at each of them. Johnson's rule constructs an optimal schedule by partitioning jobs into sets. The first set contains jobs that have processing time in the first stage lower than processing time in the second stage and the second stage vice-versa. Jobs from the first set are scheduled first in increasing order of their processing time (SPT). After that, jobs from the second set are scheduled in decreasing order of their processing time (LPT). Jobs with equal processing time in both stages can be added to either of these two sets. More about Johnson's rule can be found again in Scheduling by M. Pinedo [12].

In our problem of item expedition, the processing time at a scanner is always the same and is lower than processing time at a machine. This gives us an idea that SPT rule should utilize machines in FFL well. Guinet [7] proposed a heuristic method based on Johnson's rule for FFL and made a comparison to SPT and LPT rules and surprisingly concluded that LPT heuristics gives good result for makespan minimization (as stated by H. R. Kia [8]).

3.4 Local search techniques

In addition to mentioned dispatching rules like SPT, LPT or Johnson's rule, and general approaches like CP, another popular methods for solving scheduling problems are various local search techniques.

Local search is another very wide topic. In this thesis, we will describe the very basic idea behind local search algorithms. Again, detailed information and examples can be found in Scheduling by M. Pinedo [12].

The local search algorithm takes an existing schedule an tries to improve it by searching through his neighborhood. The neighborhood consists of schedules that are generated from the original schedule using some specific operation; we will be referring to it as move. The algorithm makes a step by evaluating schedules from a neighborhood and selecting new schedule from which it creates a new neighborhood and begins another step. There are many types of local search algorithms that usually differ in the way they select or reject the next move or how they create neighborhoods.

The specific algorithms that are often mentioned in the literature are tabu-search, simulated annealing, and genetic local search.

Local search methods also do not guarantee to find an optimal schedule.

3.5 Robustness

In real-world, many unpredictable events might occur and disrupt our schedule. How to reduce the effect that disruption has on a schedule typically depends on the environment in which the schedule is used. It is often hard to say what attributes of the schedule indicate that it is robust. M. Pinedo in book Scheduling [12] proposes that insertion of idle times, avoiding job postponing and schedule less flexible jobs first.

In some cases, it may be necessary to re-schedule jobs. For example, if any machines break, jobs that were originally scheduled on that machines need to be re-scheduled. This leads to a suggestion to make scheduler fast to be able to properly react events such as machine breakdowns.

Chapter 4

Proposed solution

The proposed algorithm is designed in a way so that it is relatively simple and fast so that it can be easily applicable in practice and can be potentially used for frequent rescheduling as a reaction to some random events.

It consists of four different phases.

1. Truck allocation
2. Job dispatching
3. Re-ordering
4. Idle time insertion

The proposed method does not try to utilize stacker cranes optimally. Finding an optimal schedule requires lots of computational power. Several MIP formulations have been proposed, and various branch and bound approaches have been developed for these problems, but still, only problems with around 50 jobs can be solved in reasonable time [12]. Furthermore, because of production handling, the original schedule may be disrupted, and the value of finding an optimal schedule will be lost.

That being said, in the first phase algorithm optimally distributes trucks to ramps, so that load on these ramps is equal as possible and thus solves the problem of truck allocation (Subsection 2.1). In the second phase, however, heuristic construction is used to get a good schedule by solving item dispatching problem (Subsection 2.2) in a relatively short time. The third phase is used to further improve on assignments, which were made in the second phase using local search algorithms. In the last phase, we insert idle times in between completion times c_i to achieve good robustness to delays caused by prioritized production handling [12]. In Figure 4.1, we can see the steps except re-ordering illustrated for better understanding.

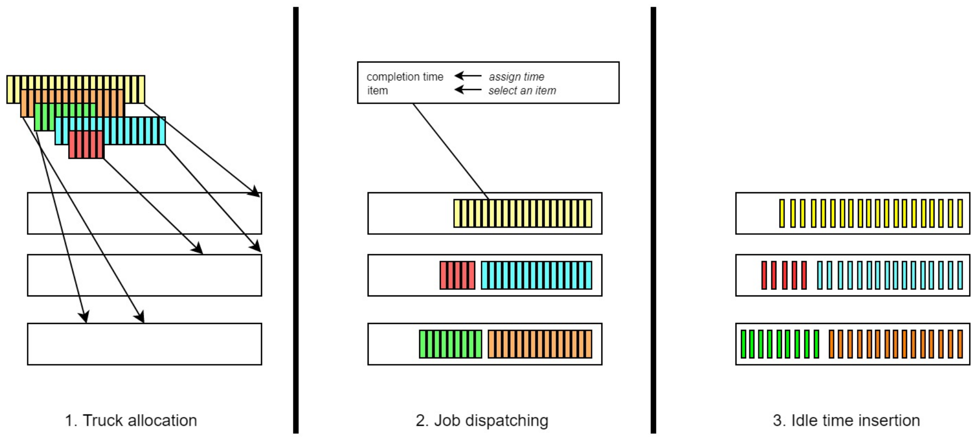

Figure 4.1: Illustration of all phases except the re-ordering phase of the algorithm.

4.1 Score of the schedule

To be able to compare different feasible schedules (See Subsection 2.2), we introduce a score of a schedule.

We formulate it as a sum of the weighted sub-objectives. The higher the score is, the better the schedule is.

Score:

$$score = -\alpha T + \beta R - \gamma U - \delta S \quad \alpha, \beta, \gamma, \delta > 0 \tag{4.1}$$

Sub-objectives:

1. **Maximum tardiness** (T)

$$\max_{i=0,\dots,n} (\max((c_i - end), 0) \tag{4.2}$$

The maximum tardiness represents the time by which the last item that arrived at the expedition ramp exceeded work hours. Ideally, the value of maximum tardiness is zero, meaning that the whole expedition fits into working hours.

2. **Sum of idle times** (L)

$$\sum_{i=0}^{n} w_i \cdot \theta_i \tag{4.3}$$

Maximizing the sum of idle times increases the robustness of a schedule (See [12]). Because one of the requirements is that items of the same type should be loaded in the given order, an expedited item that is followed by an item of a different type may benefit more from idle time. That is why we take weights of idle times into consideration.

3. **Constraint duration of truck expedition** (U)

$$U = \begin{cases} 1 \text{ if } c_i^{'} - c_i^{'} > \text{expedition duration of truck } i \\ 0 \text{ otherwise} \end{cases} \tag{4.4}$$

   This objective represents penalty for a truck expedition which takes too much time.

4. **Standard deviation of idle times** (S)

$$S^2 = \frac{1}{n-1} \sum_{i=1}^{n} (\theta_i - \bar{\theta})^2 \tag{4.5}$$

   The goal of this objective is to keep idle times uniformly distributed.

The score helps us decide which schedule might be better. Having a score of a schedule is also essential for local search algorithms, which we use in last phase of the proposed metho (Section 4.5).

4.2 Truck allocation

In this phase, we solve the problem from Subsection 2.1. The goal of this phase is to distribute trucks among expedition ramps so that workers at each ramp work almost the same time and do nearly the same amount of work. This can be achieved by minimizing the makespan. By minimizing the makespan, we achieve good utilization of machines or in our case expedition ramps, which leads to a balanced load on expedition ramps [12].

In many factories as well as in the automated warehouse, there are not many trucks scheduled for a single day (under 50). On top of that, the objective of this problem, minimizing a makespan, is a very simple objective. This makes this problem a good example where CP or MIP performs well. Even for it being a real-world problem, finding the optimal solution for this case is valuable since stochastic events like "need to reschedule a truck to different ramp" or "truck needs more or fewer items than it originally demanded" do not occur very often.

This problem was already formulated as an assignment problem in Subsection 2.1. This formulation can also be used for the CP problem.

To improve the performance of these methods, we need to select **upper bound** of the objective and variables. Upper bound is calculated using LPT heuristics mentioned in Chapter 3. Going back to the problem formulation in Subsection 2.1, we can not specify that the domain of the decision variables y_i is $\{0, \dots, C_{max}(LPT)\}$. The LPT heuristics can also be used as an alternative to CP or MIP solution for big instances of the problem.

There are no requirements for order at which trucks arrive, but assuming some trucks may be delayed. We propose ordering at which trucks arrive at the expedition ramp is determined by priority of a truck. A truck with higher priority is scheduled before a truck with lower priority on the same expedition ramp. If a truck is delayed, it affects trucks scheduled after it, and it creates a cascading effect, where all trucks behind the first delayed one are delayed if reserve time between them is not big enough.

The priorities should be specified by a client or assuming that trucks with more requested items have a higher chance of getting delayed. We order trucks in increasing order by number of requested items.

4.3 Job dispatching

In this phase, we solve the job dispatching problem from Subsection 2.2 and thus create a feasible schedule. To obtain a feasible schedule, a constructive heuristic method is proposed. This method assigns values to variables in a way that after assigning value to every variable, we get a feasible solution without a need to backtrack. In this phase, we also need to take into consideration some of the objectives. If machines are not utilized properly by the proposed method, the duration of the schedule might be greater than the duration of working hours or the breakdown of a machine could have worse consequences. Also, items should arrive at different expedition ramps in similar rate to ensure that truck expedition does not take significantly longer than expected. We propose heuristic construction Algorithm 1 to solve this problem.

The algorithm starts by putting items from the warehouse and times from domains C_i to an ordered queue *items* (See Subsubsection 23). Then, the algorithm selects a *ramp* and cycles through *machines* (stacker cranes) in established order until any *machine* has an item with the same type as a request next in order on a *ramp*. Then it selects the shortest completion time from domain C_i of an unload action of this item, that satisfy constraints from Subsection 2.2.

Machine and ramp selection heuristic

This first proposed heuristic deals with machine and ramp selection. The aim of this heuristic is to get good utilization of machines at any time and process an expedition in all active ramps at similar speeds. The rule that is proposed to sort machines based on the time of the last processed unloading in increasing order and prioritize the first one. Basically, the same rule is proposed for the selection of an expedition ramp, but it is ordered based on the time that ramp's latest loaded item passed a conveyor junction (the time an item left the warehouse). Ordering by a time an item is loaded is not suitable, since travel time to different expedition ramps can be different, and thus it may result in unbalanced loading rates. We will refer to this heuristic as *Latest job last* (LJL) heuristic.

Algorithm 1: Job dispatching

```
Result: A feasible schedule S
items ← sort items;
foreach $i$ in items do
    $C_i$ ← sort in increasing order $C_i$
end
for position ← 0 to $n$ do
    machines ← sort machines;
    ramp ← select a ramp;
    foreach machine in machines do
        foreach $i$ in items do
            if $m_i$ = machine then
                foreach time in $C_i$ do
                    if $S \cup \{i, time, ramp, position\}$ is feasible then
                        $pos_i$ ← position;
                        Add {i, time, ramp} to solution S;
                        items ← items \ {$i$};
                        goto mainLoopEnd;
                    end
                end
            end
        end
    end
    [mainLoopEnd]
end
```

Items ordering

After machine and ramp selection, we must select an item from the machine's aisle to be expedited. Then we filter out items with incorrect type and finally select an item using LPT dispatching rule. The LPT dispatching rule is used, because in comparison to SPT rule and to Johnson's rule it minimizes makespan the best in FFL according to A. Guinet [7] as is already said in Subsection 3.3.

In Chapter 6 we also test SPT dispatching rule to confirm that LPT rule is indeed superior.

We will refer to schedules created using the Job dispatching algorithm by name combination of item ordering and machine/ramp selection heuristic — for example, SPT-LJL or LPT-cyclic.

4.4 Re-ordering

This phase is optional, and its goal is to improve the schedule by re-ordering items that were ordered using LPT heuristics in the second phase (See

Subsubsection 23). The heuristics used in the second phase should create a good schedule, but it does not guarantee that the utilization of machines is optimal. This phase tries to bring it closer to the optimum.

The method that is proposed here is a local search algorithm (specific algorithms are compared in Chapter 6. A move of a local search algorithm swaps order (See Subsubsection 23) of two items. After positions are swapped, the job dispatching algorithm must be run again, but items should be now sorted by their assigned position pos_i and not by item ordering proposed in Subsubsection 23.

If we have a feasible schedule, the local search algorithm is a good alternative to the CP approach, or variants of a branch and bound, which do not perform well in problems of this scale. The local search ends after a certain amount of time or has no neighbor to select and returns the best schedule that was found so far. This implies that step of the algorithm is optional, and in the worst case, we get the schedule from the second phase.

4.5 Idle time insertion

The goal of this phase is to take advantage of the remaining time of working hours, assuming there is remaining time, spread the expedition throughout the working hours and as a result create a schedule that is more robust to delays.

Since order at which items are expedited is one of the requirements (subsection 1.1.2), inserting idle times in between loading (or completion times) is crucial for minimizing number of errors in ordering due to delays caused by prioritized production handling. It also reduces the chance of a scanner's buffer to overflow.

We associate each item with an idle time, which follows the arrival of the item to an expedition ramp. Other items cannot be scheduled to arrive after an item i for its duration of idle time θ_i.

The method that is proposed in this stage is hill climbing with a limit on move evaluations (HCLM). HCLM is same as hill climbing local search, but instead of finding the most valuable move across all moves, it first evaluates several moves and selects the one that improves the overall score of a schedule the most. The main benefit HCLM has over hill climbing algorithm or other some other local search algorithm is that we can set how many moves are evaluated (HCLM limit) in one step to regulate performance since job dispatching can be computationally demanding.

To properly use HCLM, we have to specify what moves can this method use. There are n moves, one for each item. Each move inserts some low amount of idle time (seconds) after arrival to an expedition ramp of an item. A move is part of a move list that is sorted in increasing order based on the size of idle time of an item that is associated with a move. HCLM evaluates several moves in this order and makes the next step; this continues until there are no moves that improve the score of the schedule.

In scenarios when there is a lot of time remaining (several hours) until the work hours end, idle time distribution may not perform well, because HCLM would have to make lots of steps to spread out the expedition throughout the working hours. To combat this performance issue, a special move or moves are added to the beginning of the move list. These moves insert idle times to all items at once and thereby avoids unnecessary evaluations after every idle time insertion.

On the other hand, if there is no remaining time until the end of the work hours, we can only look at the score and the trade-off between tardiness, T and the sum of idle times L and set proper weights for these parameters. If weights are set properly, we can insert more idle times using HCLM even if it means that items will be arriving at an expedition ramp after working hours.

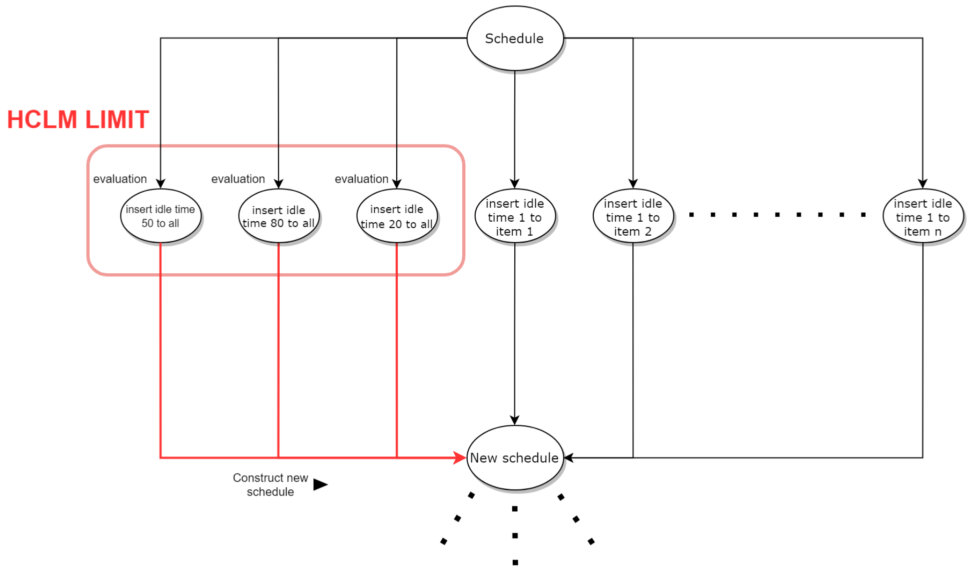
Schedule
HCLM LIMIT
evaluation
evaluation
evaluation
insert idle time 50 to all
insert idle time 80 to all
insert idle time 20 to all
insert idle time 1 to item 1
insert idle time 1 to item 2
insert idle time 1 to item n
New schedule
Construct new schedule

Figure 4.2: Diagram showing a step of HCLM algorithm.

The basic idea behind HCLM can be seen in Figure 4.2, where the first three moves are evaluated, and the best one is selected. HCLM then continues to the next step.

Chapter 5
Implementation

In this chapter, I will briefly describe the architecture of the simulation tool to describe then, how is the scheduler implemented and how it is incorporated into the simulator.

5.1 The automated warehouse simulator

The simulator is based on the AgentPolis [2], which is an agent-based platform for modeling transportation systems with discrete-event simulation core. The simulator was created for evaluation of different automated warehouse designs and is currently still in development.

The simulator is driven by an event queue. Every simulation event is added to the queue and scheduled to be fired at a specific time. A fired event is then caught by event handlers.

The most relevant event handler for this thesis is the simulator's dispatcher. The dispatcher handles events like "an item is produced," "an item is requested at some expedition ramp" and so on. The dispatcher also creates so-called stacker crane unload (SCU) activities from entries in an expedition input file and event and stacker crane store (SCS) activities from events signalizing that an item is produced. These activities are then scheduled and executed by stacker cranes. For better understanding, the following subsection describes the most relevant parts of the simulator in more detail.

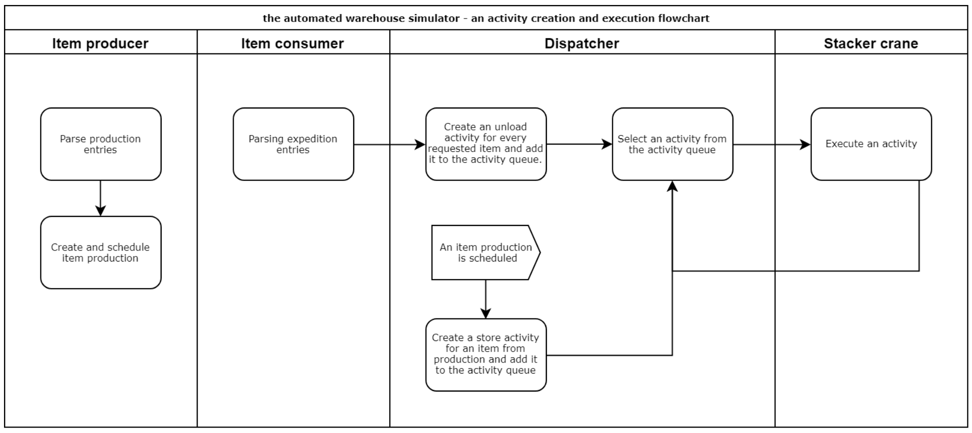
the automated warehouse simulator - an activity creation and execution flowchart
Item producer
Item consumer
Dispatcher
Stacker crane
Parse production entries
Create and schedule item production
Parsing expedition entries
Create an unload activity for every requested item and add it to the activity queue.
Select an activity from the activity queue
Execute an activity
An item production is scheduled
Create a store activity for an item from production and add it to the activity queue

Figure 5.1: Diagram showing the flow of the activity creation and execution

5.1.1 Activities

Stacker cranes can pick up or put down an item, and move horizontally or vertically in an aisle. These actions are in the simulator wrapped into storing and unloading action.

Stacker crane store activity

Every SCS activity is associated with a stacker crane, an item to be stored and storage space to which it is going to be stored.

By executing this activity, a stacker crane carries out these actions.

1. Move to the beginning of the aisle
2. Pick up the item
3. Move to the free position
4. Store the item

Stacker crane unload activity

Every SCU activity is associated with a stacker crane and an item to be unloaded.

By executing this activity, a stacker crane carries out these actions.

1. Move to the item to be unloaded
2. Pick up the item
3. Move to the beginning of the aisle
4. Put the item on the conveyor leading to expedition ramps

The activity queues

The activity queue stores activities to be executed. There are two activity queues. The first activity queue stores SCS activities and the second stores SCU activities. Both SCU and SCS activities are placed into their queues on creation. The way that activities are taken out and executed from the queue is shown in Pseudocode 2.

Algorithm 2: Activity execution

```
1  foreach $activity \leftarrow activityQueueSCS$ do
2      if $activity.stackerCrane$ is not idle then
3          $activity.execute()$
4      end
5  end
6  foreach $activity \leftarrow activityQueueSCU$ do
7      if $activity.stackerCrane$ is not idle then
8          $activity.execute()$
9      end
10 end
```

The *Activity execution* is triggered after activities creation or an activity is executed (Shown in Figure 5.1).

5.1.2 Production and expedition handling in the simulator

First off, the input of the simulator is config file describing the layout of the warehouse and stacker cranes parameters, CSV file with item descriptors and time stamps representing the production schedule and CSV file with an item type, quantities, and truck id representing an expedition to be scheduled. To be clear what information is available, see Section 1.1.2.

Knowing what activities are and that we have files specifying production schedule and expedition demand, we describe how are these activities created and executed.

Starting with production, the schedule for the production is created at the beginning of a day in the simulation. Entries from the production input file are parsed, and $PRODUCTION\_GENERATED$ events put in the event queue. These events signalize that an item is produced and is on it's way to the automated warehouse. The dispatcher handles these events and decides to what aisle will the produced item be delivered and schedules a new event, $ITEM\_ARRIVAL$, for the time it arrives at the assigned aisle. When the dispatcher handles $ITEM\_ARRIVAL$ event, it creates SCS activity and puts it into the activity queue. How are activities executed from the activity queue is explained in Section 5.1.1.

The expedition handling works similarly, but it also has some major differences. The expedition input file is also parsed at the beginning of a day in the simulation. Then one event, $TRUCK\_ARRIVAL$, per expedition ramp is created (if there are more trucks planned for that day than expedition ramps) and they are scheduled for the beginning of the work hours.

The dispatcher creates SCU activities for each a truck expedition item request upon catching $TRUCK\_ARRIVAL$ event. After every SCU activity of a truck expedition is finished, another $TRUCK\_ARRIVAL$ event is triggered for some another planned truck for that day.

Order at which trucks are called or activities are processed is random.

The simulator generates a report at the end, which includes information about truck expedition durations, time stamps of activities, reached buffer sizes, average idle times of stacker crane, among others.

5.2 Scheduler module

In this section, we will describe integration into the simulator first and then we describe the implementation of the algorithm itself.

5.2.1 The schedule integration

To integrate the scheduler, we need to change how the expedition handling works; the way activities are executed from activity queues and introduce new parameters in activities.

Starting with activities, we add a new parameter *startTime* indicating what time should this activity be executed. Since activities of activity queues are executed in random order, we need to change how they work so that we can execute activities at $currentSimulationTime \geq startTime$. Instead of two separated queues, we introduced a single new priority queue that contains both SCU and SCS activities sorted in increasing order of their *startTime*. SCS activities have their *startTime* set to zero. How are activities executed from this queue is shown in Algorithm 3

Algorithm 3: Proposed activity execution

```
1 foreach $activity \leftarrow priorityQueue$ do
2     if $currentSimulationTime \geq activity.startTime \wedge$
       $activity.stackerCrane$ is not idle then
3         $activity.execute()$
4     end
5 end
```

Now that we specified the new approach to activity execution, we describe the new way to SCU activity creation.

The expedition input file is still parsed at the beginning of a day in the simulation. Instead of scheduling $TRUCK\_ARRIVAL$ events, we schedule the whole expedition using the proposed algorithm. The newly created schedule is then converted to activities and inserted into the priority queue. The new flowchart is shown in Figure 5.2.

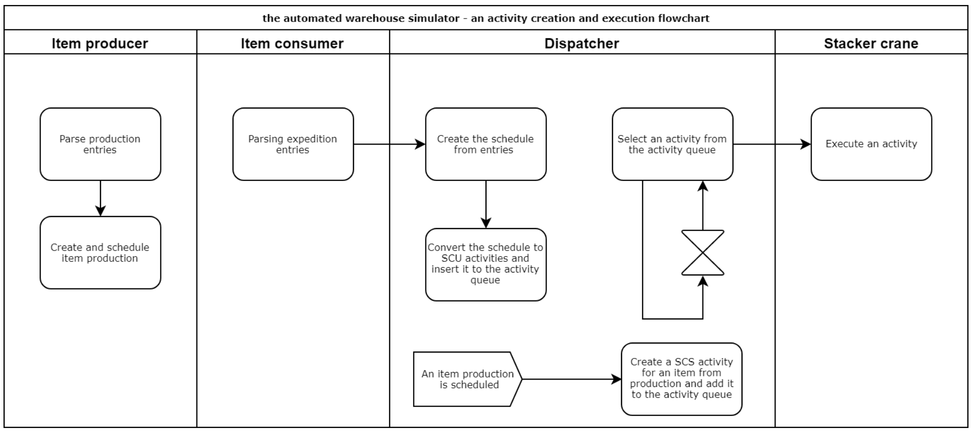
the automated warehouse simulator - an activity creation and execution flowchart
Item producer
Item consumer
Dispatcher
Stacker crane
Parse production entries
Create and schedule item production
Parsing expedition entries
Create the schedule from entries
Convert the schedule to SCU activities and insert it to the activity queue
Select an activity from the activity queue
Execute an activity
An item production is scheduled
Create a SCS activity for an item from production and add it to the activity queue

Figure 5.2: Diagram showing the flow of the activity creation and execution

The dispatcher creates SCU activities for each a truck expedition item request upon catching $TRUCK\_ARRIVAL$ event. After every SCU activity of a truck expedition is finished, another $TRUCK\_ARRIVAL$ event is trigger for some another planned truck for that day.

To be able to evaluate the schedule in the simulation, we create a new schedule from the execution and completion time of activities at the end of the simulation. We can then compare this schedule, where some activities might have been delayed to the one created by the scheduler.

5.2.2 Algorithm implementation

The following described the details of the implementation of the algorithm from Chapter 4. How the schedule is created is proposed in Chapter 4, but to achieve good maintainability and performance open-source libraries are used in the first, third and fourth phase of the algorithm. The first phase was solved using open-source constraint programming library, Choco [13]. Both third and fourth phases were solved using open-source planning engine, Optaplanner [9], which is used for the local search.

Truck allocation

As proposed in Section 4.2, truck allocation is solved using Constraint programming. Specifically, We use the Choco open-source java library dedicated to constraint programming [13]. The library can be used to model the problem in a declarative way by stating the set of constraints. The library comes up with many different constraints with state-of-the-art implementations, offers different search algorithms and strategies, and also provides us with multi-thread resolution. Because of this, the scheduler can achieve good performance and allows a developer to tweak or modify the solving process easily.

Re-ordering and Idle time insertion

To use local search in Optaplanner, we need to model the problem by specifying domains, variables, and the calculation of the score. The score consists of two long values, one for hard constraints and the other for soft constraints. Domains, variables, and hard constraint score corresponds to the problem formulation from Subsection 2.2, where for each violated constraint the hard score is subtracted by one. The soft score calculation is the same as the score calculation from Section 4.1. Another thing that is needed is to generate moves of local searches, as stated in Sections 4.4 and 4.5. Order of moves, search techniques and parameters for different local searches, like for example temperature of simulated-annealing, can be all setup in the configuration file of the Optaplanner's solver.

Chapter 6
Evaluation

In this chapter, I will introduce datasets for evaluation, and present comparison of different heuristics, local search techniques, and lastly show qualities of a schedule generated using the proposed algorithm.

Different configurations and input data are used to show examples that properly represent the results in this chapter. Layout and stacker crane configurations are based on real-world automated warehouse design. Expedition and production input data are also derived from anonymized real-world data provided by a client for a project involving the simulator.

6.1 Datasets

Overview of some of the input data and configurations can be found in following subsections. Details of used configurations and overviews of all input data can be found in Appendix A. For each following section in this chapter; there are more examples of results in other appendices. Those appendices are mentioned in corresponding sections.

6.1.1 Input expedition files A1-A4

Expedition input files A1, A2, A3, and A4, will be used to compare the performance of schedule creation. These expeditions files have a similar number of trucks, requested item types and distribution of these types, but they differ in number of items to be expedited. Attributes of these input files are shown in Table 6.1.

attribute/name	A1	A2	A3	A4
number of requests	1016	4157	8284	16568
number of trucks	10	10	10	10
number of types	84	101	101	101

Table 6.1: Overview of A1, A2, A3 and A4 input files.

6.1.2 Layouts B1-B4

Layouts B1 to B4 are used for comparison of different heuristics. We compare heuristics on these different layouts because some of the heuristics perform differently if there is a single or several expedition ramps. Also if the throughput of the warehouse is limited by stacker cranes and not scanner, we are unable to properly approximate optimal makespan. More on that in Section 6.3. We briefly describe layouts B1 to B4 in Table 6.2.

Layout	Description
B1	Has three expedition ramps and 48 stacker cranes.
B2	Has only one expedition ramps and 48 stacker cranes.
B3	Has three expedition ramps and 20 stacker cranes.
B4	Has only one expedition ramps and 62 stacker cranes.

Table 6.2: Description of layout configurations.

Detailed layout and stacker crane configurations can be found in Appendix A.

To make referring to these combination of layouts and input expedition files easier, we will name these combination *inputFileName/layoutName* — for example, A1/B1.

6.2 Performance

In this section, we simply show how long does it take to create the schedule from different expedition input files of different sizes, specifically A1/B1 - A4/B1. We take a look at phases two, three, and four since the first phase have an insignificant overall effect on execution time. For phases three and four, we measure the number of evaluated moves in half a minute (excluding runtime of phase two). The following Table 6.3 shows the results.

	A1/B1	A2/B1	A3/B1	A4/B1
phase 2	371ms	1673ms	4623ms	16820ms
phase 3 moves evaluated	417 steps	39 steps	12 steps	2 steps
phase 4 moves evaluated	330 steps	39 steps	10 steps	2 steps

Table 6.3: Performance evaluation of schedule creation.

In Table 6.3, we can see the duration of phases two to four and how much they change for datasets of different sizes. We can also see that we are not able to evaluate much re-ordering moves of the phase three in a 30-second window. We show how valuable applying re-ordering moves might be in Subsection 4.4. The same is shown for idle time insertion in Section 4.5. Although, in phase four, we can still distribute a lot of idle times using a method proposed at the end of the Section 4.5 by move or several different moves which uniformly distribute idle time to all items in a single move.

6.3 Heuristics comparison

Schedules are created using different rules and heuristics. For each schedule, we take a look at their utilization of machines by comparing the makespan of schedules created by phase two of the algorithm. We can see makespan comparisons in Figure 6.1.

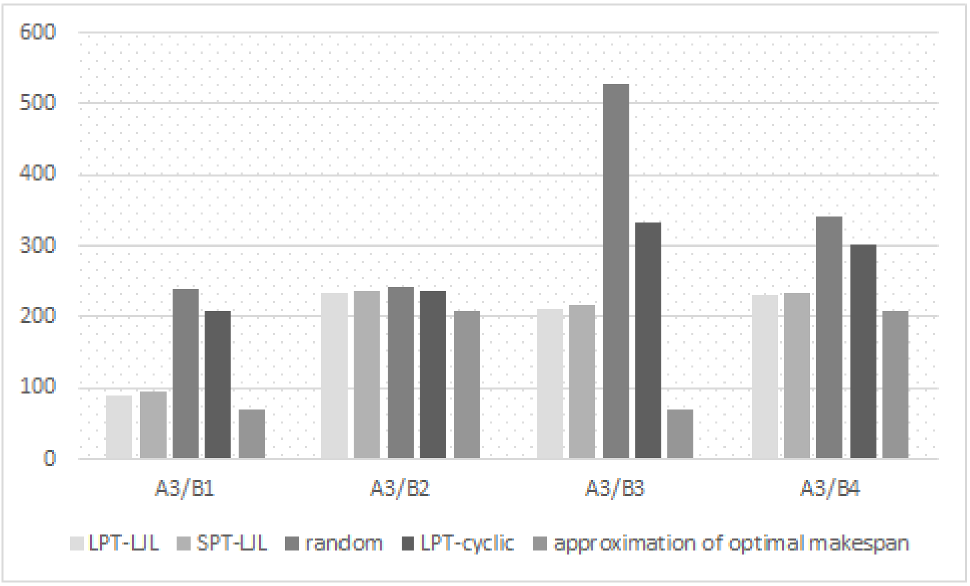

Figure 6.1: Makespans of schedules generated using different heuristics.

To be able to tell how well the machines were utilized, we also show the optimistic lower bound of optimal makespan in Figure 6.1. To obtain optimistic lower bound, we assume that throughput of the warehouse is limited only by a scanner and calculates how long does a scanner need to scan all items on the expedition ramp that expects the most items. The calculation of this approximation can be seen in Equation 6.1.

$$\max(s \cdot size_0, \dots, s \cdot size_r) \tag{6.1}$$

From results depicted in Figure 6.1, we can tell that selection heuristic can play a big role in utilization, especially if there are several ramps (Layout B2). LPT-cyclic heuristic can match LPT-LJL and SPT-LJL if there is a single ramp. LPT-LJL heuristic results in the best utilization in both layouts, although in comparison to SPT-LJL, the difference is negligible in this example.

We can see that for dataset A3/B3 the difference between the makespan approximation and other heuristic is quite big. For this dataset the number of available stacker crane is much lower than in other datasets. Because of that a scanner is no longer the limiter of throughput of the warehouse and Equation 6.1 is not suitable for makespan approximation. In average, the most utilizing heuristic, LPT-LJL, has makespan 17% bigger than the makespan approximation excluding datasets with layout B3. For comparison makespan of SPT-LJL, random and LPT-cyclic heuristics are in average 21%, 108% and 85% higher than makespan approximation respectively.

More examples can be found in Appendix B.

6.3.1 Effects of re-ordering and idle time insertion

We will now take a look at the effect of phases three (Section 4.4) and fourth (Section 6.3.1) of the proposed algorithm. We remind again that in the third phase, we use local search algorithms to improve the utilization of stacker cranes further. The fourth phase then inserts idle times between item arrivals at expedition ramps. It is expected that doing that; we will prevent some items from arriving at the expedition in the wrong order. Also, by inserting these idle times, it will make the final schedule more robust to scanner buffer overflows.

Re-ordering

We will take a look at the third phase, re-ordering. On top of comparing schedules created in the second phase to schedules from the third phase, we will also compare makespan improvements for several different local search algorithms.

The following Figure 6.2 shows makespans of LPT-LJL schedule, schedules after re-ordering using tabu, simulated-annealing, and hill climbing search for a minute.

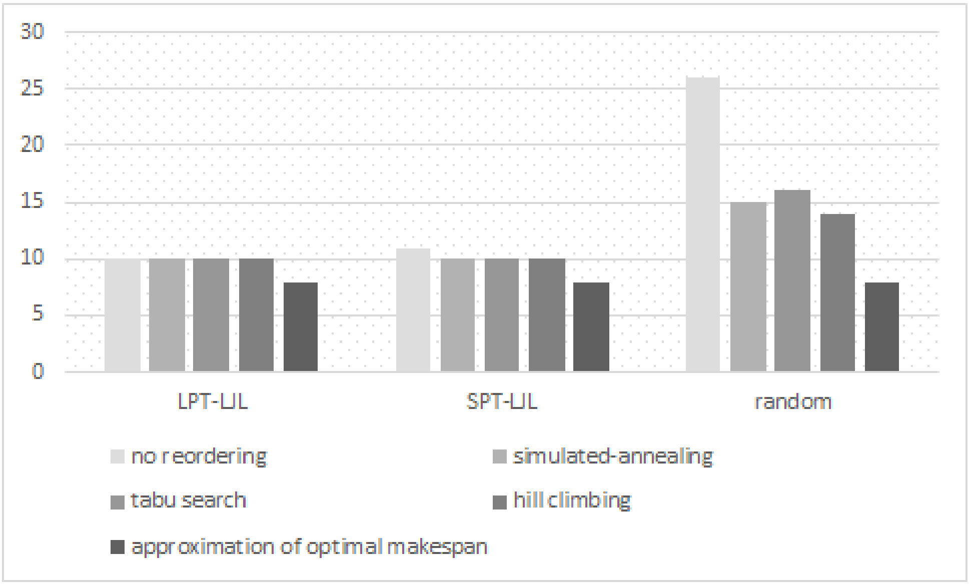

Figure 6.2: Comparison of schedule's makespans before and after applying re-ordering using different local search algorithms. The figure was generated from data A1/B1

We can see that for proposed heuristic LPT-LJL re-ordering has no effect in this case, it also has minimal impact on SPT-LJL heuristics. The more significant effect of reordering can be observed in a case, where no heuristics are applied, and makespan is then significantly reduced. More examples can be found in Appendix B and they have shown similar results.

Re-ordering is an optional phase of the algorithm and based on the results it should be only used if there is no requirement on the overall speed of the algorithm or we get new requirements from a client.

Idle time insertions

By inserting idle times, we want our schedule to be more robust to unexpected delays caused by production prioritization. Another than a score, robustness will be measured by parameters from the simulation report. These parameters are:

1. max. buffer size - Maximum number of concurrent items at any scanner buffer at any time during the simulation.
2. buffer usage - Total number of items put in a scanner buffer throughout simulation.
3. unexpected arrivals - Total number of items that arrived in an unexpected order. During scheduling, we decide on the order at which items arrive at an expedition ramp. If an item violates this order, it counts towards unexpected arrivals.
4. violated type orders - Total number of items that arrived at an expedition ramp in a wrong order. It is similar to unexpected arrivals, but the difference is that we count only items that arrived in an unexpected order and are of a different type than the expected item.

The first column of the following table 6.4 shows the score, makespan, and other attributes of a schedule generated in the second phase using LPT-LJL heuristics. The second column shows parameters of that schedule after the simulation. The third and fourth columns are also showing attributes before and after the simulation, but the schedule was created using all four phases of the proposed algorithm.

	LPT-LJL	result	final schedule	result
makespan [min]	89	126	720	720
T [s]	0	0	0	0
1. max. buffer size	0	6	0	0
2. buffer usage	0	1117	0	0
3. unexpected arrivals	0	3690	0	137
4. violated type orders	0	307	0	14

Table 6.4: Comparison of schedules before and after idle time insertions. Generated from A2/B1

First, two columns of the table show that without insertion times the expected duration of the schedule is several minutes longer than expected, items arrived at wrong order and scanner buffers have been used quite a lot too. Comparing this to the third and fourth column, we can see that the schedule has much bigger makespan, but tardiness is still kept at zero. Also, there is no difference between the real and the expected makespan in this example. But parameters that tell us the most about the robustness are buffer usage and unexpected orders. The exemplary result, Table 6.1, shows that these parameters were greatly improved, but we also conducted more experiments with different warehouse layouts, which can be found in Appendix B. Results of those experiments shown that schedule with no idle times have on average 215 times more items ending up in a scanner's buffer at some point than a schedule with idle times. We also got results that number of unexpected arrivals are three times lower in schedules with idle time insertions.

We also compared these results to the results got from schedules generated using random heuristic and idle time insertion. By comparing these results, we can tell that schedules using LPT-LJL heuristic have still perform better and that by applying idle time insertion we do not diminish the effect of phase two of the algorithm to the point it does not matter. On average schedules with idle times generated using random heuristic have 52% more items on unexpected positions and 15% more items ending up in a scanner's buffer at some point than a schedule with idle times generated from LPT-LJL heuristic.

Chapter 7
Conclusion

In this thesis, we implemented a scheduler suitable for the specific automated warehouse and its environment. The original problem was split into a truck allocation problem and the flexible flow line scheduling problem. The first problem of assigning trucks to ramps was straightforward and was solved using constraint programming. Constraint programming is seemingly the most popular method for optimizing NP-hard scheduling problems, but it requires an excessive amount of computational resources to solve real-world scheduling problems, where number of jobs is often in thousands. For this reason, we proposed a heuristic construction algorithm, which can create a feasible schedule quickly. We improved its utilization of machines by incorporating dispatching rules and heuristics based on the special cases of the FFL problem. To further improve the utilization of machines, we introduced local search methods to swap selected scheduled items and potentially correct heuristics shortcomings. Lastly, we used local search methods again, to insert idle times in between item arrivals to expedition ramps.

The proposed method with all of its steps was implemented and integrated into the simulator, where the scheduler could be properly tested. We used two open-source constraint programming libraries, Choco and Optaplanner, to keep the scheduler easily modifiable and maintainable.

Results have shown that proposed heuristic LPT-LJL utilize machines to 117% of the optimal makespan approximation. At that point, improvements in machine utilization by re-ordering was computationally heavy, and further improvement was negligible. Other heuristics utilized machines to 121%, 208%, and 185% of the optimal makespan approximation, respectively. The idle time insertion proved to be very useful in improving the schedule's robustness. Schedules with idle times had zero or on average three times fewer items arrived in a wrong order compared to schedules without idle times. Also, we have shown that schedules without idle times 43% of items had to be stored in a scanner's buffer on average. Only 0.2% of items had to be stored in a scanner's buffer on average during the execution of the generated schedule with inserted idle times.

For future work, the scheduler could be further improved by analyzing the different objectives of this problem and designing a proper genetic algorithm that can further improve existing scheduler more efficiently. That can be further extended to the area of machine learning.

Bibliography

[1] Gilgen Logistics AG. Automated high-bay warehouse. `https://commons.wikimedia.org/wiki/File:Automatisches_Hochregallager_mit_Regalbediengeräten.jpg`.

[2] aic.fel.cvut.cz. Agentpolis. `https://github.com/aicenter/agentpolis`.

[3] Krzysztof Apt. *Principles of Constraint Programming*. Cambridge University Press, 2003.

[4] Peter Brucker. *Scheduling Algorithms*. Springer-Verlag, Berlin, Heidelberg, 3rd edition, 2001.

[5] Min Dai, Dunbing Tang, Kun Zheng, and Qixiang Cai. An improved genetic-simulated annealing algorithm based on a hormone modulation mechanism for a flexible flow-shop scheduling problem. *Advances in Mechanical Engineering*, 2013, 08 2013.

[6] Jinxiang Gu, Marc Goetschalckx, and Leon F. McGinnis. Research on warehouse design and performance evaluation: A comprehensive review. *European Journal of Operational Research*, 203(3):539 – 549, 2010.

[7] A. Guinet, M.M. Solomon, P.K. Kedia, and A. Dussauchoy. A computational study of heuristics for two-stage flexible flowshops. *International Journal of Production Research*, 34(5):1399–1415, 1996.

[8] H.R. Kia, H. Davoudpour, and M. Zandieh. Scheduling a dynamic flexible flow line with sequence-dependent setup times: a simulation analysis. *International Journal of Production Research*, 48(14):4019–4042, 2010.

[9] KIE. Optaplanner. `https://www.optaplanner.org/`.

[10] Sandeep Kochhar and Robert J.T. Morris. Heuristic methods for flexible flow line scheduling. *Journal of Manufacturing Systems*, 6(4):299 – 314, 1987.

[11] R L. Graham. Bounds on multiprocessing timing anomalies. *Siam Journal on Applied Mathematics - SIAMAM*, 17, 03 1969.

[12] M. Pinedo. *Scheduling: Theory, Algorithms, and Systems.* Prentice Hall international series in industrial and systems engineering. Springer, 2008.

[13] Charles Prud'homme, Jean-Guillaume Fages, and Xavier Lorca. *Choco Documentation.* TASC - LS2N CNRS UMR 6241, COSLING S.A.S., 2017.

Appendix A

Overview of used layouts, stacker crane configuration and expedition input files

A.1 Layouts

attribute	B1	B2	B3	B4
expedition ramp count	3	1	3	1
aisle count	48	48	20	62
aisle length [m]	110	110	110	135
aisle height [m]	16	16	16	20
rows	35	35	35	50
columns	135	135	135	135
conveyor speed [m/s]	1	1	1	1

Table A.1: Overview of layouts

A.2 Stacker crane configuration

Single stacker crane configuration is used in all examples.

attribute	SC configuration
max driving speed [m/s]	4
driving acceleration [m^2/s]	1.75
vertical speed [m/s]	1
vertical acceleration [m^2/s]	1
loading speed [s]	5
unloading speed [s]	5
capacity	1

Table A.2: Stacker crane configuration

A.3 Expedition input files

attribute/name	A1	A2	A3	A4
number of requests	1016	4157	8284	16568
number of trucks	10	10	10	10
number of types	84	101	101	101

Table A.3: Overview of expedition input files.

Appendix B

Tables of comparisons schedules before and after idle time insertions

	LPT-LJL	result	final schedule	result
makespan [min]	89	126	720	720
T [s]	0	0	0	0
L [s]	0	0	104858	30511
S [s]	0	0	0.2	0.7
1. max. buffer size	0	6	0	0
2. buffer usage	0	1117	0	0
3. unexpected arrivals	0	3690	0	137
4. violated type orders	0	307	0	14

Table B.1: Before and after idle time insertion comparison. Generated from A2/B1

	random	result	final schedule	result
makespan [min]	235	241	720	720
T [s]	0	0	0	0
L [s]	0	0	88151	30153
S [s]	0	0	0.2	0.7
1. max. buffer size	0	4	0	1
2. buffer usage	0	466	0	1
3. unexpected arrivals	0	3209	0	298
4. violated type orders	0	10	0	10

Table B.2: Before and after idle time insertion comparison. Generated from A2/B1

	LPT-LJL	result	final schedule	result
makespan [min]	234	235	720	720
T [s]	0	0	0	0
L [s]	0	0	296590	29429
S [s]	0	0	0.2	0.2
1. max. buffer size	0	21	0	1
2. buffer usage	0	3044	0	11
3. unexpected arrivals	0	3177	0	988
4. violated type orders	0	241	0	49

Table B.3: Before and after idle time insertion comparison. Generated from A2/B2

	random	result	final schedule	result
makespan [min]	388	388	720	720
T [s]	0	0	0	0
L [s]	0	0	23956	28091
S [s]	0	0	0.2	6.7
1. max. buffer size	0	5	0	2
2. buffer usage	0	3044	0	18
3. unexpected arrivals	0	2685	0	1372
4. violated type orders	0	145	0	60

Table B.4: Before and after idle time insertion comparison. Generated from A2/B2

	LPT-LJL	result	final schedule	result
makespan [min]	212	564	720	830
T [s]	0	0	0	110
L [s]	0	0	296590	36651
S [s]	0	0	0.1	13
1. max. buffer size	0	2	0	2
2. buffer usage	0	98	0	22
3. unexpected arrivals	0	3194	0	1642
4. violated type orders	0	190	0	73

Table B.5: Before and after idle time insertion comparison. Generated from A2/B3

	random	result	final schedule	result
makespan [min]	543	889	720	979
T [s]	0	169	0	259
L [s]	0	0	50662	46361
S [s]	0	0	0.7	15
1. max. buffer size	0	1	0	1
2. buffer usage	0	54	0	15
3. unexpected arrivals	0	2840	0	1879
4. violated type orders	0	165	0	79

Table B.6: Before and after idle time insertion comparison. Generated from A2/B3

	LPT-LJL	result	final schedule	result
makespan [min]	220	220	720	720
T [s]	0	0	0	110
L [s]	0	0	29105	29105
S [s]	0	0	0.1	4
1. max. buffer size	0	10	0	1
2. buffer usage	0	3182	0	7
3. unexpected arrivals	0	2886	0	650
4. violated type orders	0	183	0	26

Table B.7: Before and after idle time insertion comparison. Generated from A2/B4

	random	result	final schedule	result
makespan [min]	336	336	720	720
T [s]	0	0	0	0
L [s]	0	0	27328	29512
S [s]	0	0	0.2	5
1. max. buffer size	0	7	0	1
2. buffer usage	0	1009	0	7
3. unexpected arrivals	0	2134	0	912
4. violated type orders	0	86	0	25

Table B.8: Before and after idle time insertion comparison. Generated from A2/B4

Appendix C

CD content

Root directories	Content description
sources	source codes of expedition scheduler module
data	input files and configurations for the simulation tool

Table C.1: Content description of CD