Department of Information and Computing Sciences

Machine Scheduling in Printing: a Problem from the Industry with Sequence-Dependent Setup Times and Precedence

Computing Science Master Thesis

510033

Author:
R. Ruiter

Supervisors:
Dr. J.A. Hoogeveen
Dr. ir. J.M. van den Akker
Co-supervisor:
P. de Bruin

November 2023

Chapter 1

Introduction

Suppose you have written a work that you want printed in book form. You do not want, or need, to go through a publisher, and you are not able to convince a large-scale book press to print a few dozen copies of your write-up. Luckily, you have other options. Some companies will print, not press, your book on commission.

There is a difference between the process of printing a book rather than pressing it. Book pressing uses machines that stamp the contents on a large sheet of paper, which is afterwards folded into the appropriate page composition. Book printing uses large printers, that essentially function just like your home or office printer but are built for a large throughput of pages. Printing is cheaper, but much slower than pressing, making it more suited for smaller quantities.

So, you search for one such company and place an order. There are a lot of options to make this bookwork exactly to size. Hard- or softcovers, different types of material, matte or shiny laminate, ring-binding, glue-binding... chances are one of the standard options will be good enough for you, or that you want something a bit more special.

First, you must deliver your digital files. There are some instructions for correct formatting provided on the website, but you are not that skilled in graphic design and in a few days the company sends your files back to you with a few requested changes that are necessary to make your order fit for printing. You, not confident in your skills, request that the company's 'prepress' team does this for you instead, for a fee.

A few days later you receive your files again. This time, you may find that you are unsatisfied with the changes they made, or decide that they are O.K. Either way, this communication continues until finally both you and your contact agree that the file looks good. After this, the printers finally start running, but only to provide you with a single proof copy of your future book. When you give the O.K. for this version, the company can finally start printing your full order.

Now all the components must be printed. Figure 1.1 shows a simplified production process. Both the inner work and the cover must come off the printers. It is likely that your inner work is completely in black and white, and your cover is in colour, but perhaps you got creative and made every page in your book a different colour. After this, the cover is laminated, with the matte, shiny or scratch-free finish of your choice. If you requested as such, a hard cover is made to fasten the cover around. Now that all the pieces are together, they can be joined together with the binding method that you preferred.

At least, that is how the process goes at the company Ipskamp Printing. This is one such company that takes private orders as well as business orders for printing books in small amounts (say, 20 to 2000 books per order). In this thesis, we will model their production process to create an automated planning system through machine scheduling theory.

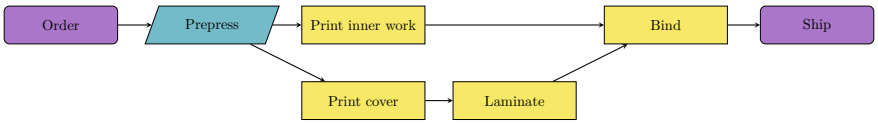

Figure 1.1: Simplified flowchart of the production process

Chapter 2

Analysis

We begin our analysis by examining the areas where Ipskamp Printing's scheduling system can be enhanced, focusing on the company's current order processing decisions. They currently employ a digital order system that only allows for the order of printing to be scheduled, and do not schedule finishing and binding. They adhere to a convention of printing on as little materials (paper of different weights per kilogram, wood containing or woodfree, cream or white) per printer as possible on one day. The schedule exclusively includes orders for printing, with other production steps being addressed by employees as they arise.

The printing operations are invariably allocated to the fastest printer that is able to print the necessary colours (i.e., black-and-white pages will never go on the colour printer). However, the long-term scheduling consequences of always picking the printer in this way are not considered, even if this leads to other printers idling unnecessarily.

Predominantly, errors and unexpected situations that occur are from order discrepancies rather than machine malfunction. The files might not be fit for the printer after all, or the customers want to make some last-minute changes to their order. When the latter occurs, usually the order is taken out of the schedule until contact with the customer has been established and agreements about changes have been made. The consequence for the rest of the schedule is that the rest of the jobs for that day are moved forward. When possible, a different order with the same material is moved up to go on the printer for that day, but when this is not possible this simply means idle time.

In the case that a printer is malfunctioning, there is usually an alternative printer available to take care of the task. However, the printers all vary in speed and capabilities. The company usually takes this into account and will for example look for space on the monochrome only printers for a monochrome only printing task, as to not waste time and money on the costly to operate full-colour printers.

Delving further into the details of the operations that must be performed for the jobs, the printing step of the production process starts with slotting the necessary material into the printer. Since changing of material takes a while, the planning managers who manually decide when to print orders aim to minimize the amount of time lost on this process by always planning for a single material per printer per day, as mentioned before. For laminating, material must also be changed to match the order specifications. The person working on the laminating station decides in which order to laminate the products up for that day and usually groups the covers by material as to minimize time lost to changing material. For all other operations, there is a standard setup time per job; this entails moving the product to the machine and loading the pallet in. Because of this, negligible differences exist for the size of the order, but we choose to assume this is constant.

The company reports that their storage is consistently overflowing. Currently, the schedule does not consider the storage consequences. This storage issue occurs with half-products stored between the operations as well as finished products that are stored until the established delivery

day. Frequently the half-products are covers or inner work for which the other part of the product has not been printed yet. These problems are most likely consequences of making the scheduling decision for printing the cover and the inner work independently of each other. However, the company also reports that storing finished products takes up a lot of their floorspace. They report that sending out orders earlier is not an option, since the customers usually arrange their availability to receive the order.

Certain printers use paper on a roll rather than individual sheets. These are a special case since their replacing is more difficult and time-consuming than the sheets, but also since a roll of paper will run out after a few hours of printing. For the purpose of our project, we will assume that the roll has infinite capacity, since it is difficult to continuously log the amount of paper used and available on a more detailed scale than amount of rolls available. For this and other types of paper used, we assume that the stock of materials is infinite. If a type of paper for an order is not available, we will consider it an order-based malfunction, which we will talk more about further in this paper.

Currently, the schedules are created by hand. The planner uses a program called 'multipress' that automatically drops tasks at the appropriate printers on the day that the order is finalized. Then, the planner places the tasks on days according to his discretion. He considers the size of the order, the deadline, which other materials are printed that day, how complex the product is, and who the customer is. He creates an idealized version of the schedule but works with the assumption that at the end of the day the schedule will not be entirely finished, due to lack of time, materials or machine breakdown.

The company reports a concern that the planner must spend a lot of time moving the unfinished tasks of the day to the following day. They would like this to be automated. We will consider unfinished tasks from the previous days in arranging our schedules. However, that this is a major issue tells us that the created schedule for the day is almost never finished. Our scheduling efforts will naturally be focused on creating schedules that can get finished in time as often as possible, and that can adjust to the situation when machine breakdown happens.

2.1 Optimization Framework

2.1.1 Minimization Objectives

In optimizing Ipskamp Printing's scheduling system, we aim to minimize the following key objectives:

1. Total machine idle time between tasks.
2. Frequency of material change within a day.
3. Instances of storage overflow.

Out of these objectives, our primary focus will be on minimizing the idle time between tasks, followed by addressing storage overflow concerns, given the reported issues with storage capacity.

2.1.2 Factors

To achieve these objectives, we will consider several factors that play a critical role in scheduling decisions:

1. *Machine Preferences:* Tasks must be carried out on the appropriate machine.
2. *Materials:* Group jobs by material to minimize setup times between different materials.
3. *Storage:* Limit instances of storage overflowing.

4. *Deadline Restrictions:* Finish the order close to the deadline. Not too far ahead of the deadline, since the customer is expecting the shipment on a fixed day.

5. *Error Handling:* Ensure enough time between order steps to account for errors.

It is important to note that some of these factors may conflict with our minimization objectives. For example, reducing idle time might require allocating buffer time for error handling, which could extend production schedules.

2.1.3 Assumptions

In the formulation of our scheduling approach, we make the following assumptions:

1. *Paper Roll Assumption:* Treat printers with paper rolls as having indefinite roll capacity.

2. *Material Stock Assumption:* Assume unlimited material stock.

3. *Operator Assumption:* Assume there are always enough operators available to finish the proposed schedule.

These assumptions provide a basis for setting up the scheduling framework, with the understanding that real-world constraints may require adjustments as necessary during implementation.

Chapter 3

Modeling the problem

Having assessed Ipskamp Printing's scheduling challenges and potential improvements, we should find a theoretical framework that can accurately model our situation. First, we must determine what kind of scheduling problem we are dealing with. Since we are planning the production process around machines, we should look at machine scheduling theory for answers. For us, each order or 'job' is a series of operations that must be executed on several machines.

3.1 Flow shop

The classification that suits this kind of problem best is the 'flow shop', also named the multistage problem (Johnson, 1954).

3.1.1 Classic Flow Shop Characteristics

The classic flow shop problem is characterized as follows:

- There are m machines and n jobs.
- Each job J_j contains m operations.
- The i-th stage of a job must be executed on the i-th machine.
- No machine can execute more than one operation simultaneously.
- Operations within jobs have a specified sequence, but jobs can finish in any order.

3.1.2 Deviations from Classic Flow Shop

1. **Objective Function**: In the classic scenario, the objective function is to minimize the maximum end time of the schedule. Our objectives are much more complex than that, and therefore we can not use this same objective function.
2. **Sequence Dependent Setup Times**: There is another trait of our problem that is different from the classic flow shop. The operations need to be set up, and in the case of a printing operation, the setup time can either be changing the roll of paper in the printer or nothing, when the correct material is already in the printer. Therefore, the setup time is dependent on which material the operation was using that came before. This means that our flow shop model has 'sequence dependent setup times'. Each job J_i is assigned an integer and positive *processing time* p_i and an integer and positive *setup time* s_{ij} that is required if job J_j is immediately executed after job J_i. The problem with sequence dependent setup times has been proven to be equivalent to the traveling salesman problem and hence NP-hard (Pinedo and Hadavi, 1992).

3. **Variability in Operations Count**: Diverging from the classic flow shop model, our jobs can have varying numbers of tasks. This means not all jobs will have m operations. To address this we simply consider all reduntant operations to have a processing time of 0.

4. **Parallel Execution of Operations**: While the classic flow shop model has tasks executed in sequence, certain jobs can occur concurrently in our model. For instance, the cover and inner work of the book can be printed independently, and both have the binding step as a successor task.

5. **Multiple Possible Machines**: While the classic flow shop model has a set machine for each task, in our case some jobs have multiple options for machines. Between these options, there are differences in efficiency.

3.2 Error handling

It is possible for unexpected situations to occur on an already established schedule. In the following sections we will explain different scenarios in which the schedule might have to react to unexpected changes.

As previously mentioned, most errors result in the order's removal from the schedule until contact with the customer is established. For the purposes of this thesis, we will let machine breakdown lead to the tasks being pushed ahead in the schedule. It has to be noted that automatically re-assigning orders from a paper roll-based printer to a sheet-based printer is somewhat problematic, since the sheet printer needs to select a paper based on the dimensions of the order, while the roll-based printer does not and rather folds and cuts the order into the correct dimensions.

The other kind of unexpected change is the possibility of a rush order from a company, who wants their order finished as fast as possible. We count this as an error, since it has a similar effect to an order having to be reprinted; it means that the rest of the orders will be delayed. Currently they simply push other orders out to accomodate for carrying out this order. If these orders are placed the day before, they can simply be placed in the schedule during the nightly scheduling. However, if they are placed the same day, there is no time to carry out a full rescheduling. For these situations, we ensure that there is enough room in the schedule to take these orders as they come.

3.2.1 Rolling Horizon Implementation

To address these fluctuations, we will implement a rolling horizon. What that means is that each night the schedule for the next two workweeks is established. For an overwhelming majority of the orders, at the time when the order is known and finalized, the deadline of that order is within this scope, and in many cases much shorter.

This constant flow of incoming orders will affect the previously established schedules. As we establish the schedule for the next two workweeks, it will need to change quite often to accomodate new orders. While the schedule for the second workweek is set with certain expectations at the time of its creation, it is a given considering the pattern of order placement that the second half of this schedule will experience major changes before that part of the schedule is reached.

Regardless, we still schedule with a rolling horizon of two weeks ahead, since larger orders are usually finalized and announced longer before the deadline. This means that the full picture of what needs to be done for these orders is known much further ahead. Consequently, the tasks associated with these larger orders often need to start well before their respective deadlines and thus scheduling for them should start longer before their deadline.

An error in the printing process usually means that the task will have to be attempted again later. For this reason, we spread out the tasks within a job according to complexity. We define

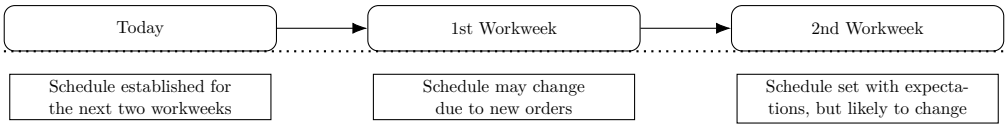

Figure 3.1: Illustration of the Rolling Horizon Implementation.

complexity as the amount of tasks required to finish the product. This way, we make sure that there is enough time left to fit the larger jobs with more tasks in, while not wasting storage space on smaller quantity orders that can be quickly corrected in case of error.

It is possible to switch printer to a different, slower one in case one of the printers breaks down. However, the finishing stations are all a single machine, and therefore if one breaks down orders will have to be outsourced or orders will have to be delayed. The company reports that they prioritize finishing smaller orders first, as that takes pressure off the amounts of emails and calls that need to be sent. However, implementing an error reporting system for unavailable machines is beyond our scope.

3.3 Preemption

Preemption is not usually desirable in the schedule, since in most cases it causes difficulty for managing the storage. For instance, half-products belonging to the same order might be inadvertently placed in different locations, causing machine operators to be uncertain about the completeness of the half-products they handle, leading to potential miscounts in printed quantities. However, there are also printing jobs that take more than two hours, and in some cases, even up to six hours. Their extended duration complicates scheduling, since they take up so much space that other orders that are, for example, same-day rush orders or announced a day ahead, are no longer able to hit their deadline. We work around this by breaking tasks longer than an hour up in single hour blocks that share all the same data, including the same previous and next events, and letting the schedule work with those tasks. This way, a large order can be broken up over time and not get in the way of the rest of the schedule.

3.4 Storage

In our analysis in Section 2 we briefly discussed the reported issues with storage overflow. We have 2 different solutions for solving this problem with regard to half-products and finished products. For minimizing the number of finished products left in storage we considered incorporating an earliness penalty into the objective function. The intent was to guide the model towards a schedule where jobs would finish as close to the deadline as possible, to ensure that orders remain in storage for only a short moment before shipping. However, upon building the scheduling system, we found that the way we dealt with half-products already covered the problem of finishing the products close enough to the deadline.

We manage storage by employing a template for optimizing job distribution and implementing a penalty system for exceeding certain product-in-process thresholds. While the former measure does not directly use the current amount of product in storage, it has a great effect on ensuring that half-products are not left in storage when this is not necessary. Larger orders necessitate more time in storage, since the risk of losing time to error is higher. Due to this, the solution is guided towards leaving a lot of time between tasks in a large order.

The former measure relies on a direct awareness of how many products are currently in storage. We expand on this measure further in Section 8. The intention is to provide steps for overflowing storage with a penalty proportional to severity. A high amount of product in

storage is not considered a problem as long as it facilitates reaching the order deadlines. However, once the upper limit of storage is approached, navigating the storage becomes difficult for the employees and the overflowing storage will hinder the flow of orders. Because of this, we have penalties that are active at set intervals. It must be noted that there is a difference between the volume individual books take up based on their size, cover type and page count, but we consider this difference negligible.

Given that the company has not reported any issues with operator availability, our scheduler assumes that an operator is always on hand when a task is scheduled for a machine. It is important to note that operators typically avoid leaving a printer idle if a task is available, regardless of if the task is scheduled to be delayed. Therefore, the model will aim for as little idle time as possible so that theory can match practice as closely as possible.

3.5 Time

In our scheduling approach, we treat each day as having unlimited time. This makes us able to ensure that tasks are always planned within their deadlines. The schedule will accommodate the necessary overtime, effectively treating the day as if it has no end.

This method assumes that the combined workload over the week will not result in an excessive amount of overtime. Should this assumption prove inaccurate, we'll need to modify the schedule, potentially allowing tasks to extend beyond their deadlines. As we implement this scheduling approach, we'll closely monitor its alignment with real-world working hours and adherence to deadlines.

Machines	
Name	Sequence-dependent setup times
Prostream printer	Yes
Image press printer	Yes
i300 printer	Yes
6250 printer	Yes
iGen printer	Yes
Laminating	Yes
Sewing	No
Sealing	No
Binder	No
Quickbinder	No
Wire-O	No

Table 3.1: Available machines and whether the setup-times are sequence-dependent

3.6 Machines

We are working with the machines outlined in Table 3.1. There are a few ways to finish a product. Printing the bookwork and the cover, and laminating the cover is always necessary. After this, a product might need to be sewn and bound, only bound, or will be bound with Wire-O wires.

3.6.1 Finishing Steps

The amount of finishing steps to a product, as well as the quantity to be made, contribute to the overall complexity of the product and therefore how much time between steps is required.

3.6.2 Material Setup Times

Some of these machines are printers that can only handle certain kinds of material or sizes. In these cases, switching between materials takes setup time. That means that when you have multiple products that use the same material, there is no setup time between these products. We schedule with the objective to group printing tasks that use the same type of paper together to reduce setup times.

3.6.3 Material Clustering

An initial problem we were running in to while implementing was that days tend to get different clusters of the same materials. Certain common materials might have a group on the start of the day and the end of the day, while other materials are stuck in-between.

Non-stream printers are easier to load, since paper can be loaded in while the printer is running, but for consistency and because material is usually clustered together regardless, we will ignore the difference between stream and non-stream printers.

3.7 Initial schedule

In the initial schedule, we schedule all tasks to be completed on the day before the job's deadline. This is already the right place for a reasonable volume of the jobs, namely print proofs and other products with a quantity below 10. For other jobs, with higher complexity or quantity, this is not the right place, and it will be left to our scheduling algorithm to find a new distribution of the tasks.

3.8 Outsourcing

It is possible for an order to be fully or partially outsourced. In the case of a fully outsourced order, we do not need to plan it, since the time the order is finished is then fully dependent on the schedule of the company where it is outsourced.

Partially outsourced orders are a little more complicated. This could mean that in the middle of the production process, the schedule has to account for the order leaving and returning to the company. Since the shipping out and receiving back times are dependent on a different company, these provided dates will be used and not changed for the schedule. They are instead provided with the correct prerequisites for the production step that they represent.

Chapter 4

Literature Review

4.1 Flow Shop Scheduling

4.1.1 Johnson's multi-stage scheduling

Johnson (1954) pioneered flow shop scheduling, then calling it multi-stage scheduling (Johnson, 1954). In this paper, he considers two and three stages. In the simple example of 2 machines A, B he gives, he can find the optimal sequence with regards to minimizing the makespan with a simple decision making scheme that runs in time complexity $O(n\,log\,n)$. The same two-stage problem with sequence-dependent setup time was proven to be NP-complete by a reduction from the Traveling Salesman Problem (Gupta and Darrow, 1986).

4.2 Industry research

4.2.1 Paints and plastic

Further industry research also covers parallel flow-shop problems with sequence dependent setup times, as it is a quite realistic case description for many real-life industrial settings not just limited to the printing industry. Theoretical cases have been based on, for example, factories where colours need to be changed out of a vat before switching to a different colour in a paint factory or in plastic manufacturing where paints have to be changed depending on the order (França et al., 1996, Conway et al., 1971). Such problems are also common in the glass blowing industry, where the type of glass must be changed out (Radhakrishnan and Ventura, 2000).

4.2.2 Textiles

A paper by Dearing and Henderson (1984) also discusses assigning looms in the textile industry with sequence-dependent setup times like the kind we are dealing with. In their case, the parallel machines are identical (Dearing and Henderson, 1984). Our printers are not identical and have different printing times for each order, but we may take inspiration from textile industry problems as they are more extensively researched than printing industry problems.

One of the earlier textile factory problems has been considered by Salvador (1973), which draws from an application in the synthetic fibres industry. In this paper, the scheduling problem is presented as a network, which allows no in-process inventory and does not consider setup times. França et al. (1996) expand on applying scheduling problems to textile factories by introducing sequence-dependent setup times, basing themselves on a factory where the loom must be replaced with a specific kind of warp chain depending on the material that is used. In their optimization function they incorporate early/tardiness penalties, as well as use in-process inventory. They work towards minimizing lost production capacity due to changeovers. This paper uses a Tabu Search algorithm, which is a search method with several similarities to Simulated Annealing that

explores the solution space by moving to its neighbours and when it discovers a local minimum it jumps to a non-improving solution to search its neighborhood as well.

4.2.3 Circuitry

Another paper that delves into the nuances of sequence-dependent setup times in an industrial setting is presented by Schaller et al. (2000). Their study focuses on the practical application of printing circuit boards, which are categorized into families. Within these families, transitions between jobs demand minimal setup. In contrast, switching from one job family to another requires a more extended setup period.

The researchers approached this challenge as a pure flow-shop scheduling problem, which is inherently "strongly NP-hard." Given the complex nature of the problem, they embarked on developing heuristic algorithms. They also explored and modified existing optimization algorithms from the 2-machine problem to cater to the specific demands of sequence-dependent setup times. Notably, they chose to iterate upon algorithmic solutions for non NP-hard versions of the 2-machine problem.

Two explicitly featured scheduling rules that they put to the test are the Composite Dispatching Rule (CDS) that performs the best for scheduling jobs within each family, and Nawaz-Enscore-Ham (NEH) which is the best for minimizing overall makespan.

In conclusion, Schaller et al. (2000) provide valuable insights into sequence-dependent setup times. Their work stands out not only for its application in the domain of industrial circuit board production but also for its intricate examination of the challenges posed by the NP-hard nature of the scheduling problem and their efforts to find polynomial solutions.

Their solution is framed as a pure flow-shop scheduling problem, editing existing optimization algorithms for the 2-machine problem to suit a sequence dependent setup time problem. They iterate upon algorithmic solutions for non NP-hard versions of the 2-machine problem.

4.3 Scheduling Techniques

4.3.1 Improvements with simulated annealing

Zegordi et al. (1995) made use of simulated annealing techniques in solving an early-tardy job shop scheduling problem. To supplement their model, they used a 'Backward-Forward Exchange Priority Table'. This means that the algorithm, before starting a swap, analyses individual tasks in a selection based on their position, duration, and other weights, and decides through this which tasks are the best candidates for a swap. This prevents wasting iterations on swaps that can be identified as unacceptable beforehand, but makes the solution somewhat rigid.

Radhakrishnan and Ventura (2000) use Simulated Annealing to solve a problem with sequence-dependent setup times with earliness-tardiness in the objective function. Naderi et al. (2009) use simulated annealing for sequence dependent setup time problems as well. This paper applies statistical research that suggests that SA is not outperformed by genetic algorithms and improves on the heuristic by integrating NEH into the method (Nawaz et al., 1983).

4.4 Addressing disruptions and changes

4.4.1 Machine vs order disruptions

Katragjini et al. (2013) consider different possible types of malfunctions including machine breakdowns, but also less researched issues like order disruptions. Machine breakdown means that a machine can be unexpectedly out of order for a known (due to scheduled maintenance) or unknown amount of time. Order disruptions are separate from machine breakdowns in that an order disruption means that the order cannot be executed as scheduled for one reason or another.

Wang and Choi (2012) describe several types of approaches for breakdown handling: robustness, reactive, and predictive-reactive. In robustness breakdown handling, jobs are planned with an additional amount of slack time. Usually, workers of the company in question independently decide to print another order when there's time left at the end of the day. Reactive scheduling makes decisions for each free moment without scheduling ahead at all.

Katragjini et al. (2013) implemented four different disruption recovery routines: a simple repair mechanism, a local search procedure of one single pass based on the insertion neighborhood, an iterated local search until a local optimum, and the Iterated Greedy algorithm by Ruiz and Stützle (2008). Upon each disruption, all four methods are triggered simultaneously, with the new schedule's score evaluated to determine the best-performing approach. They found that inserting a single local search step for each insertion significantly improves the standard iterated greedy algorithm.

A notable drawback of these methods is their myopic nature: they often overlook pertinent information that might emerge on the production floor, such as errors and rush orders. Addressing this limitation, Fabrycky and Shamblin (1966) proposed a probability-based dynamic queue discipline rooted in due dates. This ensures that orders due soon are less likely to be rescheduled.

4.4.2 Predictive scheduling

Scheduling for a certain amount of time ahead is called predictive scheduling. In fact, there are usually no pure predictive scheduling problems in industry since the chance of failure is always present. The company in question has reported dealing with faulty orders quite frequently and having to react its scheduling to that. The scope for pure mathematical programming is quite limited because of the varied nature of real-life industry- there is only so much one can predict and schedule ahead of time. One of the earlier works identifying and tackling this problem is the SONIA knowledge based scheduling system by Collinot et al. (1988). They built a predictive schedule with reactive components that can respond to deviations to the schedule coming from the production floor. Matsuura et al. (1993) introduced a method for predictive scheduling that switches to reactive scheduling when an uncertainty threshold is met, which is where Wang and Choi (2012) based their method on. They proposed integrating this method into a model additionally inspired by Lawrence and Sewell (1997). Complete rescheduling is not practical due to processing time and increased uncertainty in the scheduling being a detriment to working efficiently. Simply delaying the entire schedule keeps the most consistency but is also prone to introducing costs that can be avoided by partially rescheduling.

4.4.3 Buffer management

With regards to limiting the number of orders in throughflow, we can look towards scheduling literature on 'limited buffers'. Hakimzadeh Abyaneh and Zandieh (2011) have worked on a flow shop model that implements different buffers for each machine stage. They used a genetic algorithm to solve their model. Chou et al. (2008) implements a technique that may be interesting for minimizing the amount of storage used. The paper by Chou et al. has total weighted completion time as an objective function, which is the completion time weighted against the size of the orde. In short this means that in their solution minimizing the completion time takes priority.

4.5 Initial solution strategies

4.5.1 Ordering according to Ruiz

Research has emphasized the importance of a sound initial solution (Ruiz and Stützle, 2008). This paper cites Naderi et al. (2009) for creating the NEH-heuristic which prioritizes jobs with the highest processing times. However, we are aware that ordering jobs in this way will likely lead to a lot of setup time in the solution. Furthermore, for printing jobs the processing time depends on which printer is selected, but we already know that it is more likely to be efficient to print on a printer with a low processing time on this job. Therefore, we propose that our initial solution will first prioritize sorting printing jobs by material, and then prioritize earliest deadline first.

4.5.2 Hybrid strategies

A paper by Jin et al. (2006) considers two strategies for the initial solution of a hybrid flowshop, both of which are based on prioritizing the job with the largest processing time. These methods are FAM (First Available Machine) and LFM (Last Free Machine). The former means that a new job will be assigned to the machine that first finished the last job assigned to it. The latter means that a new job will be assigned to the last machine that becomes available, until that machine's schedule is full, and it becomes unavailable. The rationale of the LFM method is to improve the future assignment of the currently unscheduled jobs when two or more machines provide the same length of partial schedules. Jin et al. found no significant difference in efficiency between the two methods.

4.6 Simulated annealing

4.6.1 Foundations

Using simulated annealing as a base for solving optimization problems has been proposed by Kirkpatrick et al. (1983), as the method for simulating crystal behaviour as had been defined by Metropolis et al. (1953). Simulated annealing draws an analogy between minimum-energy states in a physical system and finding a minimum cost solution. This is as a liquid crystallizing: the particles of a liquid are arranged randomly. If a liquid crystallizes and is cooled too quickly, it can take on errant shapes. Similarly, in a simulated annealing algorithm the algorithm starts 'hot', the schedule is randomly rearranged, and a wide array of possible solutions are accepted. As the algorithm 'cools', the changes become smaller and the requirements for an accepted solution become tighter. While the algorithm judges new solutions, a solution with a better score is naturally accepted. However, a solution with a worse score may be accepted with a certain probability. This way, the algorithm prevents itself from getting stuck in local minima. In the algorithm the temperature may be adjusted and tuned as well as the speed at which the temperature decreases such that the algorithm can produce good solutions in a reasonable span of time.

4.6.2 Theoretical framework

A theoretical framework for using the simulated annealing method based on the approach by Kirkpatrick et al. was defined by Romeo and Sangiovanni-Vincentelli (1991). They define the model as a discrete inhomogeneous Markov chain, which is a stochastic process defined on a countable state space, with a probability measure defined on the set of transitions between states. This method has been picked up for use in job scheduling literature by Van Laarhoven et al. (1992), to find near-optimal solutions on job scheduling problems. It is also called probabilistic

hill-climbing, to move away from the metal analogy. For our purposes we will however continue to call it simulated annealing.

The Markov chain property of simulated annealing means that the algorithm moves through states at distinct intervals of time. Through these visits, it can revisit past states or move on to a new one. The new destination depends only on the algorithm's current location, as the Markov property means that the future is conditionally independent of the past.

4.6.3 Genetic algorithms

Several of the papers referenced so far have used genetic algorithms. However, for our purposes, we will use simulated annealing, since a genetic algorithm with as many parameters as our problem has is likely to have a very high calculation time and several papers have already proven that methods based on simulated annealing with proper steps taken to prevent getting stuck in a local minimum outperform genetic algorithms (Mirsanei et al., 2010, Naderi et al., 2008).

4.7 Resource allocation and bin packing

Masson et al., 2013 handle resource allocation with respect to servers as a bin packing problem. This is a problem that also deals with assigning tasks in such a way that the workload is evenly distributed, while respecting the total capacity.

4.7.1 Simulated annealing and bin packing

Simulated annealing is occasionally combined with bin packing as well, like when Brusco et al. (1997) use simulated annealing and attempts to use the least bins possible. This is not entirely relevant for us since we actually prefer to put machines in action. Much more relevant for scheduling is a paper by Polyakovskiy and M'Hallah (2021), who uses bin packing and just in time machine scheduling for furniture manufacturing.

Chapter 5

Methodological choices

The biggest challenge is to distill the problems of Ipskamp Printing down to a generalized scheduling problem that can be compared to the general scheduling solutions that exist in the literature. There is no literature available that is directly transferable to Ipskamp Printing their problems, but we may take inspiration from papers previously discussed.

So far, we have seen several methods for the building of the schedule, as well as methods to consider for handling errors, sequence dependent setup-times, variations between jobs, and storage. In the rest of this section we will discuss these methods again in short and in relation to the problems of Ipskamp Printing in order to try to build up the solution.

5.1 Building an initial solution

There are multiple ways to begin an initial solution, and the strategy used matters for the final model. Since the shape of our problem is rather unique to the printing industry, there is no initial solution model previously researched that fits exactly. To promote a schedule that mimics the form of schedule that the company has been previously familiar with, the job dispatch queue will be first sorted on material before it is ordered by earliest deadline first. After this, a strategy must be used in assigning to printers.

We will build our initial solution by creating a schedule where every job is scheduled a day before the deadline. For our schedule, it is known a long way ahead of time which orders are coming up. For our problem is that while the company officially promises a delivery time of one workweek after the order has been made official, order requests may be in the system far ahead of time. There is a large degree of uncertainty considering orders that are in the system this way, since it is unclear if the prepress process will succeed quickly or slowly, but the order will be expected somewhere in the coming weeks.

An additional challenge is that not every printing job has the same processing time on each printer. Further experimentation with the model is required to conclude whether to completely prioritize lowest processing time, least setup time or to take inspiration from FAM or LFM. Since Jin et al. found no significant difference in efficiency between the two scheduling methods, and this difference in scheduling priority is not time-consuming to implement, both methods will be attempted in tandem with other priorities that the initial solution should have.

5.2 Differentiating methods

So far, we have dedicated a lot of attention to the paper by Schaller, Gupta and Vakharia. The scheduling problem that they tackled does seem a lot like our problem- certain families of product have a negligible setup time between each task. However, they are working with a scenario where there are no deadlines. This makes their approach to handling job families not entirely transferable to our problem, since each job can have a different deadline in our case.

5.3 Semi-infinite scheduling

We have decided to approach the time limits of our schedule in a 'semi-infinite' way. What this means is that, while normally an infinite or an 'endless' schedule would contain a single continuum of tasks, and a non-infinite schedule time constraints of some way, we have decided to make every day infinite, while still differentiating between days.

This is useful for us since the desired finishing point of every order is highly dependent on the deadline. Non-infinite, restricted hours per day have led to schedules that tend to finish everything as soon as it becomes available- which leads to very front-loaded schedules considering that the orders usually only become available a few days beforehand. While a schedule using infinite days may overfill during busy periods, the infinite days leave room for the schedule to ensure that the deadline is always made, although human interference is needed when the schedule is no longer realistic.

5.4 Sequence-dependent setup times

For some machines the setup times are the same for every kind of task and therefore irrelevant for the scheduling order. For the printing machines, huge, unwieldy rolls of paper need to be picked out of storage and slotted in to the printer. It is a lot of work to change the type of paper being printed, so in all cases the company prefers to sort the workload within the same day by material type. Additionally, the material is left on the machine overnight. This means that they also prefer to start with the same material the following day when possible.

It takes a few hours of printing to finish the paper roll. For this algorithm, we consider the roll endless, since we currently have no capability to find out exactly how much paper is being used in the process of printing. However, we leave some buffer time so that pickup time for new material can be accounted for.

5.5 Outsourced tasks

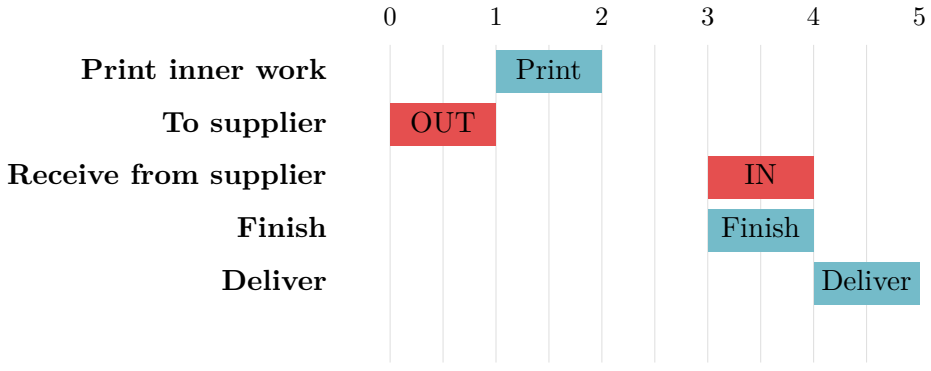

Figure 5.1: An example of a job schedule with an outsourcing

It is possible to outsource a part of the production process. This means that there is a task taken out of the schedule, and replaced with the delivery dates for the outsourced components. Figure 5.1 illustrates a scenario of a book with an outsourced task. For instance, the outsourced task might be to create a hard cover to size. Ipskamp printing does not make these covers in-house and outsources this step to a supplier. Here, the cover is custom-ordered to fit specific dimensions. OUT and IN in this case then, represent the date of ordering and receiving the product. Alternatively, the semi-finished product might be sent out for treatments that Ipskamp Printing does not have the ability or capacity to do. Such treatments might include processes

like embossing or engraving. In this case, the order is shipped out and the company must wait to receive the order back. It is important to note that Figure 4 does not specify which machine is used, as its primary focus is on the distribution of tasks.

Given that the data does not distinguish between an order dispatched for outsourcing and one awaiting an incoming product in storage, an order currently waiting for an outsourcing to finish does contribute to the total quantity of orders currently in process. This is also necessary since the business that the half-product is outsourced to, could return the product at an earlier date than expected. This way, it can be ensured that there is space for the order to arrive.

Within the software, outsourcing is depicted by two static tasks: 'in and 'out‘. They are placed in the sequence in such a way that it corresponds with the part of the process that they are replacing. In the given example, the creation of the cover is outsourced. This means that it replaces the step of printing and laminating the cover.

Chapter 6

Solution

6.1 Definitions

We begin by defining the sets involved in creating this schedule:

$$J : \text{Set of all jobs, indexed by } j.$$
$$D : \text{Set of all days, indexed by } d.$$
$$T : \text{Set of all tasks, indexed by } t.$$
$$M : \text{Set of all machines, indexed by } m.$$

Now we will continue to explain the features of each of these sets:

- **Jobs**: We denote the set of orders, henceforth referred to as jobs, as J, with n jobs represented as $J = \{J_1, J_2, ..., J_n\}$. Each job J_j has:
  - A deadline d_j
  - An order size q_j (quantity of books in order)
- **Tasks**: We denote the set of tasks as T. Each task belongs to one job at most. The set of m tasks T for job j would be $T = \{T_{j1}, T_{j2}, ..., T_{jn}\}$. Each task T_{ij} has:
  - A processing time p_{ij}
  - A target machine m_{ij}
  - A material k_{ij}
  - A set of predecessor tasks T_{prev}
  - A set of successor tasks T_{next}
- **Days**: We create a set of days D This can be an arbitrary number, but for this project we keep 10 days/2 workweeks ahead
- **Machines**: We denote the set of machines as M, with n machines represented as $M = \{M_1, M_2, ..., M_n\}$.

  Each machine $M_{s,d}$ has:
  - compatible materials k_s
  - an end time C_s

We only have one decision variable in this definition:

Decision Variables:

$x_{t,d,m}$: Binary variable. 1 if task t is assigned to machine m on day d, 0 otherwise.

And the model has several constraints to consider, in which we use $prev_task$ and $next_task$ as shorthand for the nearest precedent task and the nearest successor task:

Constraints:

1. Each task should be assigned to exactly one machine on one day:

$$\sum_{d \in D} \sum_{m \in M} x_{t,d,m} = 1 \quad \forall t \in T \tag{6.1}$$

2. Uphold task precedence:

$$\sum_{d \in D} d \cdot x_{t,d,m} \geq \sum_{d \in D} d \cdot x_{t',d,m} \quad \forall t \in T, t' \in \text{prev\_task}_t, m \in M \tag{6.2}$$

$$\sum_{d \in D} d \cdot x_{t,d,m} \leq \sum_{d \in D} d \cdot x_{t',d,m} \quad \forall t \in T, t' \in \text{next\_task}_t, m \in M \tag{6.3}$$

Task precedence relationships may be shared.

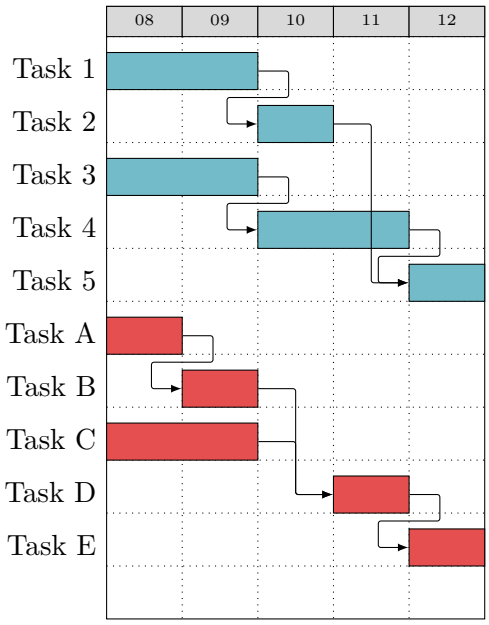

Figure 6.1: Showing shared precedence relationships

6.2 Initial solution

Initially, we schedule all order events one day prior to the deadline. For smaller orders, this approach is already optimal, eliminating the need for further adjustments. For many other orders, the intention is that the algorithm will find that moving the first task ahead will improve the score, and continue from there.

Here, "deadline" refers to the day when the order should be prepared for shipment, rather than the time it should reach the customer. The latter happens the day after this deadline. Ideally, the last operation of the order should be one day before this deadline, although completing it on the day of the deadline is also permissible.

6.3 Iterations

The standard way to run a simulated annealing-based algorithm is once for a large number of iterations. However, we have chosen to run the algorithm 10 times with a smaller amount of iterations and take the best result. This approach is adopted because the number of neighboring solutions for each schedule is relatively limited. A few thousand accepted moves in the solution is plenty to have moved every event to a different location. This means that making adjustments to a solution beyond the millionth operation is unlikely to yield significant score improvements. Multiple runs do introduce some additional overhead. However, the bulk of the scheduling time is spent iterating, and for that, multiple runs with the same amount of iterations matters very little for the total runtime.

While testing this manner of iterating, taking the average score over ten runs of the algorithm, we found an immense improvement in the average score. In the following table you can see that this improvement amounts to 40%. We suspect that the explanation for this is that the algorithm does tend towards a local minimum, but that each new iteration gives the solution a new chance to break out of it.

Average score accross 38 orders			
T value 1000×0.99			
Iterations	10 000 000	1 000 000 × 10	Improvement
Score	23044.964	13810.547	40%

Table 6.1: Improvement multi-run

6.4 Initial data

The data we work with is coming directly from the company database. Orders contain an internal calculation that dictates the components and the machines that they are to be run on. The calculation is according to the available materials and sizes that the company has. The output is a list of tasks to be carried out, connected to the machines they belong to. They have an internal calculation that gives them a time on a machine. The processing times provided per operation per printer are sourced from the suppliers of the machines, who have an estimation of amount of product that can be processed per minute.

6.5 Algorithm process

The algorithm tries to pick the day before or after the day that the task is currently on. Then, it checks if any of the predecessor tasks are on the schedule after that day. If not, the score is checked for moving the task there. If so, the algorithm attempts to move one of the predecessor tasks instead. It is focused on moving the tasks between days, since the order of tasks on a day is set.

It keeps going through available predecessor until there is one found that does not have a predecessor task, one is found that execute the move without breaking any precedence constraints, or the failsafe is hit. For any predecessor tasks on the same day, wait times are used to plan out the start times of the tasks.

If the scheduler meets a task that is fixed, it leaves it in the same place and finds a different task to manipulate. When there is a task moved, the program evaluates the score based on the schedule for that day, and the current state of the schedule of the job.

Chapter 7

Method

7.1 Execution

We will employ the Simulated Annealing algorithm to tackle this scheduling optimization problem. In the following sections we will explain how we handle the contstraints and objectives in what we have named the Adaptive Multi-Resource Production Scheduling (AMRPS) problem.

7.1.1 Scheduling

In this section, we outline how we represent schedules and the principles governing our scheduling approach:

Schedule Representation

- We determine the execution day for each task
- We group tasks that share the same raw materials and aim to execute them sequentially.
- The schedule creation process revolves around assigning tasks to specific days, while also optimizing the choice of machines.
- Tasks are assigned to machines using the variable $x_{t,d,m}$, indicating which machine (m) performs task (t) on day (d).
- The order of materials within a day on a machine is arbitrarily decided
- Tasks within a group are ordered according to the start time of their nearest precedent task. Tasks with the latest precedent task are also executed the latest.

7.1.2 Local Search

We faced the question of how to set up the local search according to our needs. Initially, we worked with a system that randomly selected a day between 5 and 0 days before the deadline to move the task to, excluding of course dates that lie in the past. However, using this method we often found the move rejected due to the relative density of a job's tasks, and thus, a high chance that a predecessor or successor task prevents a large step away from its current position.

We found that a neighboring solution was most often accepted when the task was moved one day forward or backward. Therefore, we modified the algorithm to only consider moving the task one day forward or backward as a neighboring solution. Again, this of course excludes days before the present or beyond the deadline. This lead to a higher rate of accepted solutions.

7.1.3 Algorithm scoring

The algorithm uses a minimization function to calculate the lowest score. The score is an abstraction of the amount of time used, with extra 'time' counted for bad schedule behavior. The score is calculated as follows, for an amount of jobs n in the solution:

$$\sum_{i=1}^{10} D_{si} + \sum_{j=1}^{n} J_{sj}$$

Further down this section, we will give a more detailed explanation on how the scoring is handled for every section of this algorithm. There, the scores for the machines and the jobs will be further elaborated upon. Additionally, some details will be given on what constitutes an undesirable schedule and what penalties are in place to prevent the algorithm from delivering a bad schedule.

It is most easily understood if we consider the schedule in two different pipelines. The 'day' pipeline and the 'job' pipeline. Both contribute to the total solution score, but both sides have differing, and occasionally conflicting goals.

7.1.4 Job score

The job score is designed to ensure that tasks within a job are efficiently spread out, allowing for buffer time in case of errors. This spread is based on the size of the order, with the goal of completing the job before its deadline. The score is influenced by how closely the tasks in a job adhere to an idealized template.

The table below provides a guideline for the desired distribution of tasks within a job. Each column represents a day, each row defines a size bracket for an order, and the numbers in the cells indicate the fraction of tasks from that job that should be scheduled on that day.

< 50	1			
< 100	.5	.5		
< 200	.33	.33	.33	
$>= 200$	.25	.25	.25	.25

Table 7.1: Idealized distribution of tasks within a job based on order size.

The penalty score is determined by the absolute difference between the ideal percentage of tasks and the actual percentage of tasks scheduled on a given day. The formula below calculates this score, where D is the idealized distribution for the current index of the day, J_t represents the total amount of tasks in the job, and J_{td} is the amount of Job tasks that are occurring on that day. J_{t_i} stands for the Day index of a task.

$$\text{Penalty Score} = \sum_{\min J_{t_i}}^{\max J_{t_i}} \begin{cases} |D_i - \frac{J_{td}}{J_t}| \cdot 100 & \text{if } J_t \neq 0 \\ \infty & \text{if } J_t = 0 \end{cases}$$

If J_t is 0, then that means that the Job has no Tasks and is therefore not scheduleable.

The figures below provide a visual representation of the scheduling output, illustrating the difference between a non-ideal and an ideal schedule for a job with a quantity of 1.

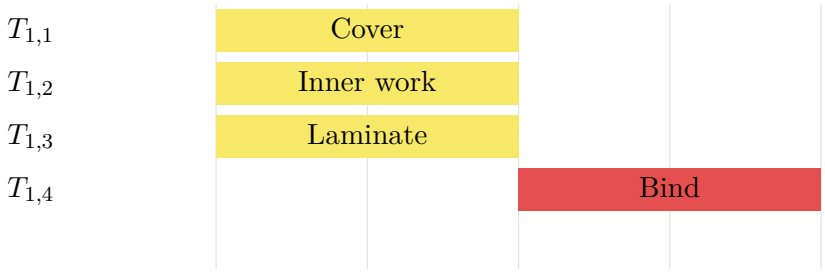

Figure 7.1: Schedule output for a quantity of 1

In this case, the penalty score is 100. Since the quantity is 1, the ideal distribution is [1], everything on one day. However, the tasks are discributed over two days, so our calculation looks like this:

$$(|(1 - \frac{3}{4})| \cdot 100) + (|(1 - \frac{1}{4})| \cdot 100) = 100$$

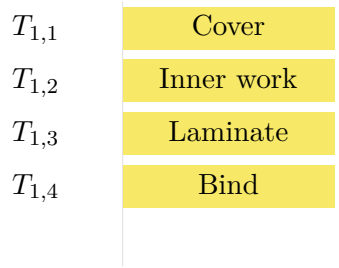

Figure 7.2: Corrected schedule output with a penalty score of 0

The table below demonstrates the calculation of the penalty score in greater detail. The calculation solely considers the absolute difference, irrespective of whether there are more or fewer tasks than ideal.

1	0	0	0	0	.25	.25	.25	0	0	.25
1	0	0	0	0	.25	.25	.25	.25	0	0
.50	0	0	0	0	0	0	0	|.25|	0	|.25|

Table 7.2: Calculating the absolute difference between the result and the idealized job throughput

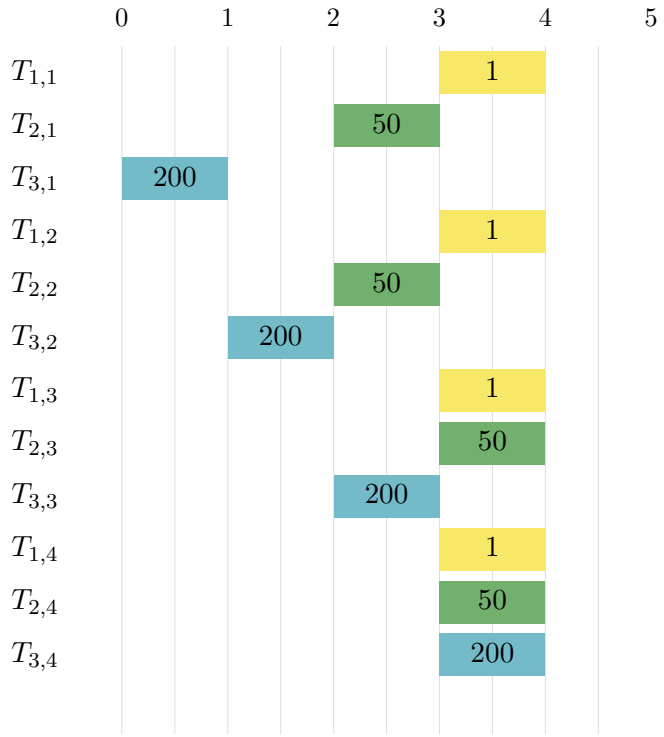

Figure 7.3: An example schedule with 3 jobs of varying quantities

Figure 7.3 shows an idealized distribution of tasks on different levels of quantities. The x-axis represents days. Each task within a job on this figure is executed on a separate machine. However, for the sake of clarity and to emphasize the spacing of tasks within each job, the machines are not represented within the chart. All the scores in this figure are 0 as they exactly mirror the prescribed arrays for the distribution of tasks for their respective quantities.

7.1.5 Day score

For the calculation of the penalty score for the day we consider three things: the content of the machines, the amount of product currently in throughput, and whether the first material that has been handled this morning matches the last material that was handled the day before on the same machine.

Quantity	Penalty
> 500	500
> 1000	1000
> 3000	3000

Table 7.3: Penalties based on quantity in production

Quantity in production We have constraints to try to control the amount of product that is currently in production. Product in production is usually busy taking up floor space, so ideally we want the jobs to be finished and out of the door somewhat quicker when it reaches a certain amount. In table 7.3 you can see the current penalties, but these are easily adjusted when the company can for example expand their storage capacity.

The total effect of this change is not very large. On a schedule example that would be considered a low amount of orders, we can see some effect in that the schedule tries to avoid

passing the 1000 number for amount in storage. In the following table, we can see one row for runs that do not have a storage penalty implemented, and one row for runs that do.

In the following table we have laid out what the effect is for applying a penalty for amount of throughput in a day, versus the effect with no penalty. The numbers in the cell represent the peak of amount of products in production at the same time accross the resulting schedule.

	Run 1	Run 2	Run 3	Run 4	Run 5	Run 6	Run 7	Run 8	Run 9	Run 10
Penalty	912	666	811	888	931	904	904	931	1039	903
No penalty	1066	760	766	1067	696	932	720	865	954	1001

Table 7.4: Highest quantity accross solution

Matching materials Now we will illustrate the penalty scores for not matching the last material from the day before to the first material for the next day. The penalty for doing this is a flat rate of 100 points. In the following tables, we define 4 successive days of production. For this part of the score we only look at the first and last material processed on that day- anything that happens in between is irrelevant for this part of the scoring. We also conveniently assume that there are only 4 machines. We take a few commonly used kinds of paper. For convenience, define it as G-Print, Silk, Recycled, Inkjet. Within these papers the grammage is relevant but for our purposes we ignore the grammage. We have $\{K_g, K_s, K_r, K_i\}$ as the possible materials.

Printer		D_1	D_2	D_3	D_4
Printer 1	Start	P_g	P_s	P_r	P_r
	End	P_i	P_s	P_i	P_i
Printer 2	Start	P_s	P_s	P_g	P_g
	End	P_s	P_g	P_g	P_g
Printer 3	Start	P_g	P_s	P_g	P_g
	End	P_g	P_s	P_s	P_i
Printer 4	Start	P_s	P_g	P_r	P_r
	End	P_i	P_i	P_r	P_i

Table 7.5: Showing the changes that would save this solution 300 points in score

7.1.6 Machine

For the machine it is the most important that the jobs are categorized by material. Therefore, we automatically keep them sorted. Within these material groups, the tasks are automatically sorted by the date and time of their last precedent task. Tasks with no preceeding task are placed first in random order. Additionally, the buffer time between each of the tasks within a material group is 5 minutes. The buffer time for switching between materials is 30 minutes.

For every machine, the start times of the tasks are calculated. The start times are always either the time the machine became available, or the time that the last precedent task ended, whichever is highest. The amount of idle time is added to the penalty score, since we want to minimize this.

We additionally have a penalty score of 100 per machine for having less than 3 different materials on any given day. This is since Ipskamp Printing had the feedback that it seemed strange to them that some days were completely empty of tasks or only contained 1 material- adding this parameter has fixed this.

7.2 Choosing the Task to move

A job is randomly selected from the list of jobs. From this job, a task is chosen. If the task has a `fixed_plan` attribute set to `true`, this iteration will stop since it is a task that has been marked as not to be moved.

7.3 Moving the Task

The method calculates the next day for the task by either incrementing or decrementing its current day. Before committing to the move, it checks if the move maintains the precedence relationship (tasks that should be executed before or after the current task). If this relationship is broken, it tries to find a different task.

7.4 Score Computation

The function then calculates the potential changes in score that would result from moving the task to a new day. This involves:

- Calculating the throughput penalty for moving the task within the job that belongs to the task.
- Assessing the consequences on the day the task is moving to. Calculating the new idle times and amount of materials on its machine on that day.
- Assessing the consequences on the day the task is moving from. Calculating the new idle times and amount of materials on its machine on that day.
- Recalculating the new start times for every task on this day on this machine
- Recalculating the new start times for every task in this job

7.4.1 Parameters

1. **Task Rearrangement (`shuffleDays`):** Initially, the algorithm suggests a new arrangement of tasks.
2. **Score Comparison:** When a valid machine is identified for a task move (i.e., when `machine` $\neq$ -1), the algorithm compares the new score to the existing score. The scoring function measures the quality of the current solution.

    - If the new score is higher than the current score, the simulated annealing calculation assesses the probability that the new solution is accepted, considering the current temperature parameter, denoted as T:

    $$P(\text{Accept}) = e^{\frac{S_i - S_j}{T}} \tag{7.1}$$

    Here, S_i represents the score of the current solution, and S_j represents the potential new score for the proposed solution.
    - Subsequently, a random number between 0 and 1 is generated. If this random number is lower than the calculated acceptance probability ($P(\text{Accept})$), the new solution is accepted, allowing for exploration of suboptimal solutions.
    - If the new score is lower than the current score, the move is unconditionally accepted, as it represents an improvement.

3. **Temperature Reduction:** The algorithm performs temperature reduction every 10,000 iterations. The temperature parameter T is reduced by a factor of 0.99, which controls the exploration-exploitation balance.

Algorithm 1 Iteration procedure

```
calculateScore(s) begin
    Initialize c, T, next_t, p, s  while c < 10000000 do
        current_score ← s.score  if new_score > current_score and changed_day ≥ 0 then
            p ← SimulatedAnnealing.getProbability(current_score, new_score, T)  if p >
            rnd.NextDouble() then
                MoveTask()  s.score ← new_score
        else if changed_day ≥ 0 then
            MoveTask()  s.score ← new_score
        next_t ← next_t - 1  if next_t == 0 then
            T ← T ×0.99 next_t← 10000
    return s
MoveTask()  begin
    Move task to a different day
```

Algorithm 2 Score calculation procedure

```
shuffleDays()  begin
    if days is not empty then
        Initialize move_score, rnd, rand_job, next, rand_index, next_index, target_task  if
        target_task.fixed_plan then
            return Error signal
        Check if moving task maintains precedence relationship  if move not allowed then
            return Error signal
        Various calculations to get penalty from machines and jobs
        return tuple containing move details
    return Error signal
```

7.4.2 Neighborhood

In order to illustrate how the algorithm computes the solution score and identifies a potential neighboring solution, we present a small-scale example. Table 7.6 lists the representative tasks, denoted as $T_{j,s}$, where j denotes the job J that the task belongs to, and s denotes the task's position within the job's sequence. For the purpose of this example, the precedence relationships entail that all jobs must be executed sequentially, meaning that tasks 1 through 4 within the same job must be carried out in order. Furthermore, we represent the duration of each task as a decimal value as number of hours (0.5 equals 30 minutes, 0.25 equals 15 minutes, and so on). Additionally, we provide the quantity of products to be printed with each job. We also provide the material for where the tasks are on sequence-dependent machines.

Task	Duration	Quantity	Material	Machine
$T_{1,1}$	0.01	1	x	1
$T_{1,2}$	0.01	1	1	2
$T_{1,3}$	0.01	1	1	3
$T_{1,4}$	0.01	1	x	4
$T_{2,1}$	0.2	99	x	1
$T_{2,2}$	0.5	99	1	2
$T_{2,3}$	0.2	99	2	3
$T_{2,4}$	0.2	99	x	4
$T_{3,1}$	0.4	200	x	1
$T_{3,2}$	1	200	2	2
$T_{3,3}$	0.5	200	2	3
$T_{3,4}$	0.4	200	x	4

Table 7.6: Duration, quantity and machine of tasks

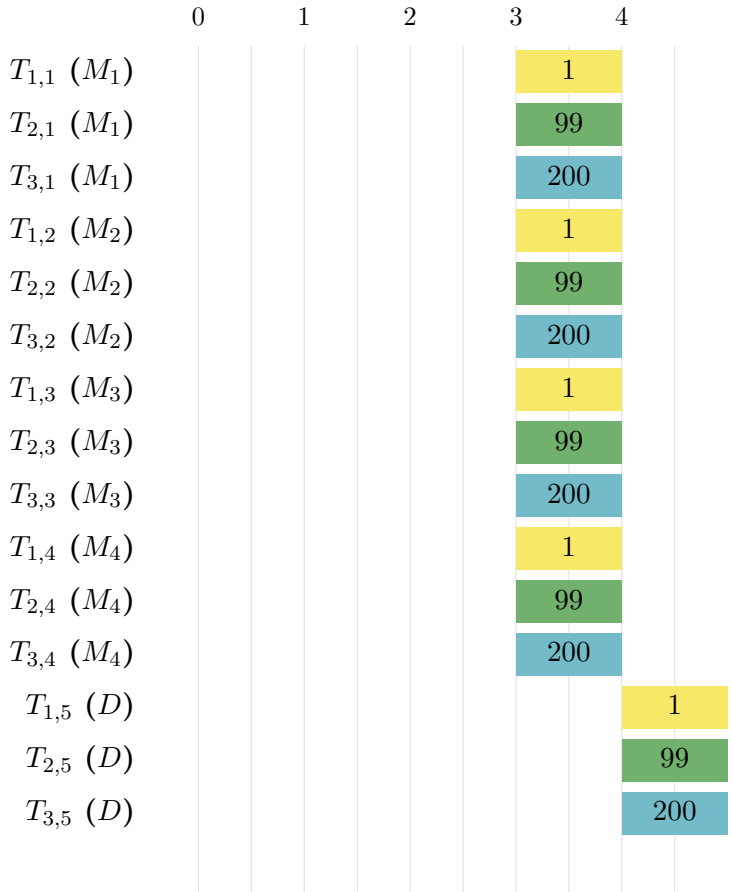

Figure 7.4: An initial schedule

Job 1's idealized array is represented as [1]. Similarly, for Job 2, it takes $[0.5, 0.5]$ units of time for its two tasks (T_2), while Job 3 requires $[0.25, 0.25, 0.25, 0.25]$ units of time for its four tasks (T_3). We calculate the job scores for Job 2 and Job 3, resulting in penalty scores of 100 and 150, respectively. Job 1 does not incur any penalty score.

Machine 1: The initial task ($T_{1,1}$) starts at time 0 and requires a duration of 0.01. Then, task 1 of job 2 ($T_{2,1}$) can start at 0.01. As it takes 0.2 units of time, $T_{2,1}$ ends at 0.21. Then,

Machine	Sequence-dependent?
M_1	No
M_2	Yes
M_3	Yes
M_4	No

Table 7.7: Machine information

$T_{3,1}$ starts at 0.21 and ends at 0.61.

- $T_{1,1}$ starts at 0 and ends at 0.01.
- $T_{2,1}$ starts at 0.01 and ends at 0.21.
- $T_{3,1}$ starts at 0.21 and ends at 0.61.

Total time spent on tasks: 0.01 + 0.2 + 0.4 = 0.61
No idle time on this machine.

Machine 2: Task 2 of job 1 ($T_{1,2}$) cannot begin at time 0 since its prerequisite, task 1 of job 1 ($T_{1,1}$), ends only at 0.01. Additionally, Machine 2 requires 0.25 units of setup time. Therefore, $T_{1,2}$ starts at 0.25 and ends at 0.26. Task 2 of Job 2 ($T_{2,2}$) must wait for the completion of $T_{1,2}$ as well as $T_{2,1}$. $T_{1,2}$ ends after $T_{2,1}$, and $T_{1,2}$ uses the same material as $T_{2,2}$, so $T_{2,2}$ can start when $T_{1,2}$ ends at 0.26, and ends at 0.76. Task 2 of Job 3 ($T_{3,2}$) has a setup time since the used material is not the same as the material for $T_{2,2}$. Therefore, $T_{3,2}$ starts at 1.01 and ends at 2.01.

- $T_{1,2}$ starts at 0.25 and ends at 0.26.
- $T_{2,2}$ starts at 0.26 and ends at 0.76.
- $T_{3,2}$ starts at 1.01 and ends at 2.01.

Total time spent on tasks: 0.01 + 0.5 + 1.0 = 1.51

- Setup time for $T_{1,2}$: 0.25 - 0.01 = 0.24
- Setup time between $T_{2,2}$ and $T_{3,2}$: 1.01 - 0.76 = 0.25

Total idle time on Machine 2: 0.24 + 0.0 + 0.25 = 0.49

Machine 3: On Machine 3 there are setup times, therefore the third operation of Job 1 ($T_{1,3}$) starts at 0.25. Subsequently, the third operation of Job 2 ($T_{2,3}$) also requires a setup time. Therefore, $T_{2,3}$ starts at 0.51 and ends at 0.71. $T_{3,3}$ has the same material as $T_{2,3}$ and therefore requires no setup time. However, $T_{3,2}$ ends at 2.01. Therefore, $T_{3,3}$ can only start at 2.01, and will then end at 2.51.

- $T_{1,3}$ starts at 0.25 and ends at 0.26.
- $T_{2,3}$ starts at 0.51 and ends at 0.71.
- $T_{3,3}$ starts at 2.01 and ends at 2.51.

Total time spent on tasks: 0.01 + 0.2 + 0.5 = 0.71

- Setup time between $T_{1,3}$ and $T_{2,3}$: 0.51 - 0.26 = 0.25

Total idle time on Machine 3: 0.25

Machine 4: The initial task of Job 1 ($T_{1,4}$) on Machine 4 begins at 0.03. Task 4 of Job 2 ($T_{2,4}$) must wait for $T_{2,3}$ and can only start at 0.71, and ends at 0.91. Then, $T_{3,4}$ relies on the completion of $T_{3,3}$, which finishes at 2.51. Machine 4 then finally ends at 2.91.

- $T_{1,4}$ starts at 0.03 and ends at 0.04.
- $T_{2,4}$ starts at 0.71 and ends at 0.91.
- $T_{3,4}$ starts at 2.51 and ends at 2.91.

Total time spent on tasks: 0.01 + 0.2 + 0.4 = 0.61

- Waiting for $T_{2,43}$: 0.71 - 0.26 = 0.45
- Waiting for $T_{3,4}$: 2.51 - 0.91 = 1.60

Total idle time on Machine 4: 0.45 + 1.60 = 2.05

Machine	**Task duration**	**Idle time**
1	0.61	0.0
2	1.51	0.49
3	0.71	0.25
4	0.61	2.05
Total	3.23	2.0

Table 7.8: Idle Time Calculation

In table 7.8 we can see that the idle time penalty for this schedule is now $3.23 \cdot 100 = 323$. This, in addition for the scores for penalty score for the Job distribution, makes a penalty score of $323 + 100 + 150 = 573$.

Task	Machine	Start time	End time
$T_{1,1}$	M_1	0	0.01
$T_{1,2}$	M_2	0.01	0.02
$T_{1,3}$	M_3	0.02	0.03
$T_{1,4}$	M_4	0.03	0.04
$T_{2,1}$	M_1	0.01	0.21
$T_{2,2}$	M_2	0.26	0.76
$T_{2,3}$	M_3	0.51	0.71
$T_{2,4}$	M_4	0.71	0.91
$T_{3,1}$	M_1	0.21	0.71
$T_{3,2}$	M_2	1.01	2.01
$T_{3,3}$	M_3	2.01	2.51
$T_{3,4}$	M_4	2.51	2.91

Table 7.9: Start times

Now we move a task to a different day according to the algorithm: say we test trying to move $T_{2,1}$ one day backwards.

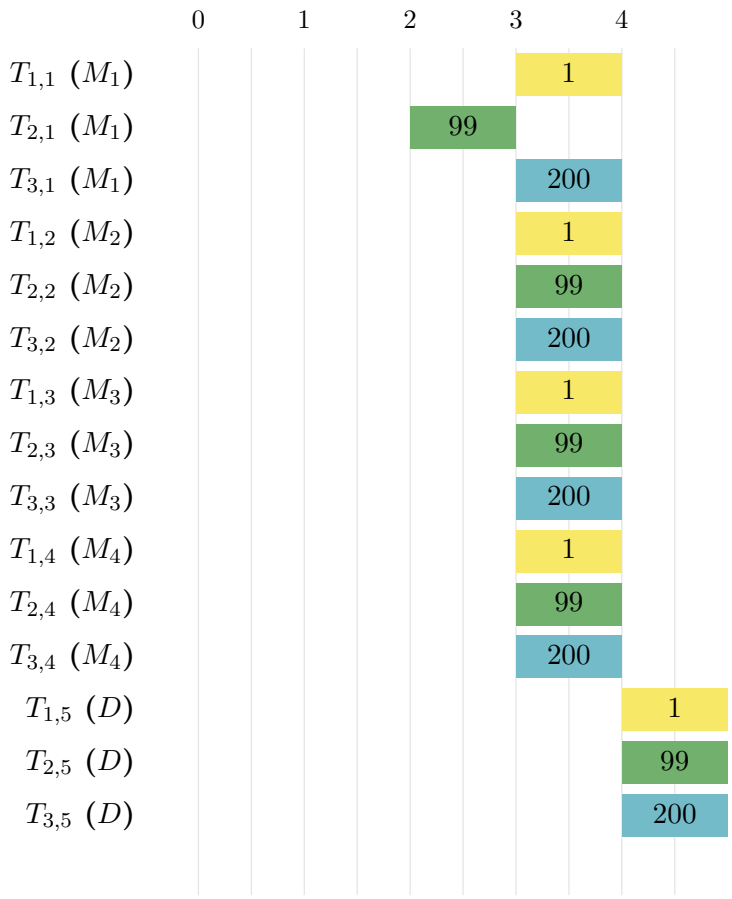

Figure 7.5: One iteration. Now, the score for Job 2 is 50 rather than 100. The penalty score of this job has been reduced by 50 due to this change.

From this we can calculate the idle times again and compare it to the previous solution.

Machine 1:

- Total time spent on tasks: 0.01 + 0.4 = 0.41
- No idle time on this machine (unchanged).

Machine 2:

- Total time spent on tasks: 0.01 (from $T_{1,2}$) + 0.5 (from $T_{2,2}$) + 1.0 (from $T_{3,2}$) = 1.51
- Idle time:
  - Setup time for $T_{1,2}$: 0.25 - 0.01 = 0.24 (unchanged)
  - Setup time between $T_{2,2}$ and $T_{3,2}$: 1.01 - 0.76 = 0.25 (unchanged)
- Total idle time on Machine 2: 0.24 + 0.0 + 0.25 = 0.49 (unchanged).

Machine 3:

- Total time spent on tasks: 0.01 (from $T_{1,3}$) + 0.2 (from $T_{2,3}$) + 0.5 (from $T_{3,3}$) = 0.71
- Idle time:
  - Setup time between $T_{1,3}$ and $T_{2,3}$: 0.51 - 0.26 = 0.25 (unchanged)
- Total idle time on Machine 3: 0.25 (unchanged).

Machine 4:

- Total time spent on tasks: 0.01 (from $T_{1,4}$) + 0.2 (from $T_{2,4}$) + 0.4 (from $T_{3,4}$) = 0.61
- Idle time:
  - Waiting for $T_{2,4}$: 0.71 - 0.26 = 0.45
  - Waiting for $T_{3,4}$: 2.51 - 0.91 = 1.60
- Total idle time on Machine 4: 0.45 + 1.60 = 2.05 (unchanged).

This means that the idle times are unchanged. Therefore, the total penalty score for the job distribution has been decreased with no penalty. This means that this new solution will be accepted.

Chapter 8

Conclusion and future work

8.1 Solution summary

In short, we have solved the Adaptive Multi-Resource Production Scheduling (AMRPS). Our solution contains a collection of jobs that contain tasks, and days that contain machines that simultaneously contain those tasks. For our current solution, those days are always ten, representing two workweeks. The start and end time of each task is documented and based on their prerequisite task and the other tasks on the machine.

We aim to distribute the workload as evenly as possible, although many other factors are in play and therefore this is not always possible. One of the factors that often prevents this is that the jobs should also be finished as closely to their deadline as possible, while still keeping some buffer time into account for handling errors. In practice, this usually means finishing the job one day before it is supposed to be sent out for shipping. We also try to keep some buffer time between tasks within the same job, depending on the size of the order with a larger buffer for a larger order.

Another important factor to them is how many times you change the material in a day. One of the printers is based on roll printing and the rolls are difficult to remove from storage and install. This is why additionally the company would prefer to start with the same material they ended with the previous day. Sequence dependent setup times would add additional time, but we also add penalty cost for the material changes because it's considered a waste of time and resources to do when it is not necessary.

Creating this solution as 'semi-infinite' was instrumental in making a maximally flexible schedule that can handle frequent underfull-ness. In the future, we may fine-tune so that the length of the schedule stays within reasonable bounds and there is some possibility for jobs to move past their deadline, but some form of semi-infinite days will remain.

8.2 Future plans

We have a few ambitions that were not yet within the scope of this project, but we would like to implement for future usage as we keep working with Ipskamp Printing:

8.2.1 Flexible printer usage

Additionally, the schedule may incorporate penalty costs for running a non-ideal printer, since certain printers are more costly to run than others. This is because certain printers are faster in the case of a large order, or faster in the case of a small order. Currently they are handling fixed parameters that decide which printer is to be used, but this model will attempt to make dynamic decisions depending on the cost of the printer compared to the resulting storage space used and with respect to the deadline of the order. This means that on occasion an order may

be printed on a printer that is not ideal for the order amount or size, but when weighed against the delay and storage it would bring to plan this order at another time, will be printed on this printer anyway.

The goal is that this schedule is implemented in such a way that the planner at the company can move towards management by exception. For this a few additional steps will be implemented to the schedule. We intend in further research to make the decisions for the binder and the printer more dynamically in accordance with what the company is comfortable with.

By that we mean that instead of rigid rules for which printer and task should go on, the program will weigh the additional time and cost of running the task on a less efficient printer, against the schedule benefits from moving the task there. Additionally, we want to implement the ability to put orders on the alternative binding machine when there is space left.

8.2.2 Incremented rolling horizon

Further experimentation is needed to analyse the impact of a rolling horizon on the working schedule. We theorize that it will balance long-term scheduling against expected variations in the schedule. Since there are quite a lot of options for materials, we consider it unlikely that a 3-day schedule ahead would be practical, but since some business orders will expect their order in a short time, scheduling very long ahead every time would also be counterproductive due to constant dynamic changes. Additionally, we will aim for dynamic changes not to be applied the same day that the order is known, since the operators tend to check the schedule for the day in the morning and cannot be expected to continuously keep checking for changes. However, we will test both scenarios where same-day changes are and are not permitted.

8.2.3 Error handling

The company has reported that they, when the need arises to push orders past their deadlines, prefer to prioritize finishing the smallest order first. This way, there are only a few contacts they need to call to report a problem with their order and work out a solution. We would like to implement a system that is aware of a priority list for orders and can behave accordingly when errors arise.

8.2.4 Data concerns

We will directly connect the printer data through JMF, which can keep track of which orders have been printed and which can directly report disturbances. From this, we can make decisions about moving tasks to different printers, and also gather more realistic data on the time that an order is in production.

8.3 Application

Tackling the challenges of creating a scheduling system that will be used in industry has been a great way to learn how to practically apply knowledge about scheduling systems. The project will now move on to its practical testing phase where the company will use this scheduling system along with their (manual) previous scheduling system. We are excited for the new technological additions that will make it easier to keep status on the machines and build a schedule accordingly.

Chapter 9

Addendum

The API connects to a frontend planning screen. Any changes in the frontend are saved directly to the database and the schedule is not reworked according to moving planning parts. This is because if someone wants to manually adjust the schedule, they likely want to adjust the schedule according to an outside force not implemented in the scheduler itself.

The scheduling interface has moveable blocks that the planner is allowed to manipulate. A block, representing a task, can also be named 'on hold' and therefore be skipped during the planning process.

Entire blocks of materials can be moved together. Movement by hand can violate the pre-requisites or the deadline, the program assumes the mover knows what they are doing.

The interface is connected to the database and shows the planned events. Both the scheduling algorithm and the UI write to the database.

This system respects the edits made to the frontend interface. There might be unexpected machine downtime, urgent orders, or material shortages. In these instances, the planner want the schedule to reflect your manual adjustments without an algorithm second-guessing their decisions.

This scheduling UI is graphically pleasant and lets the planner see in one view what the coming week of orders looks like.

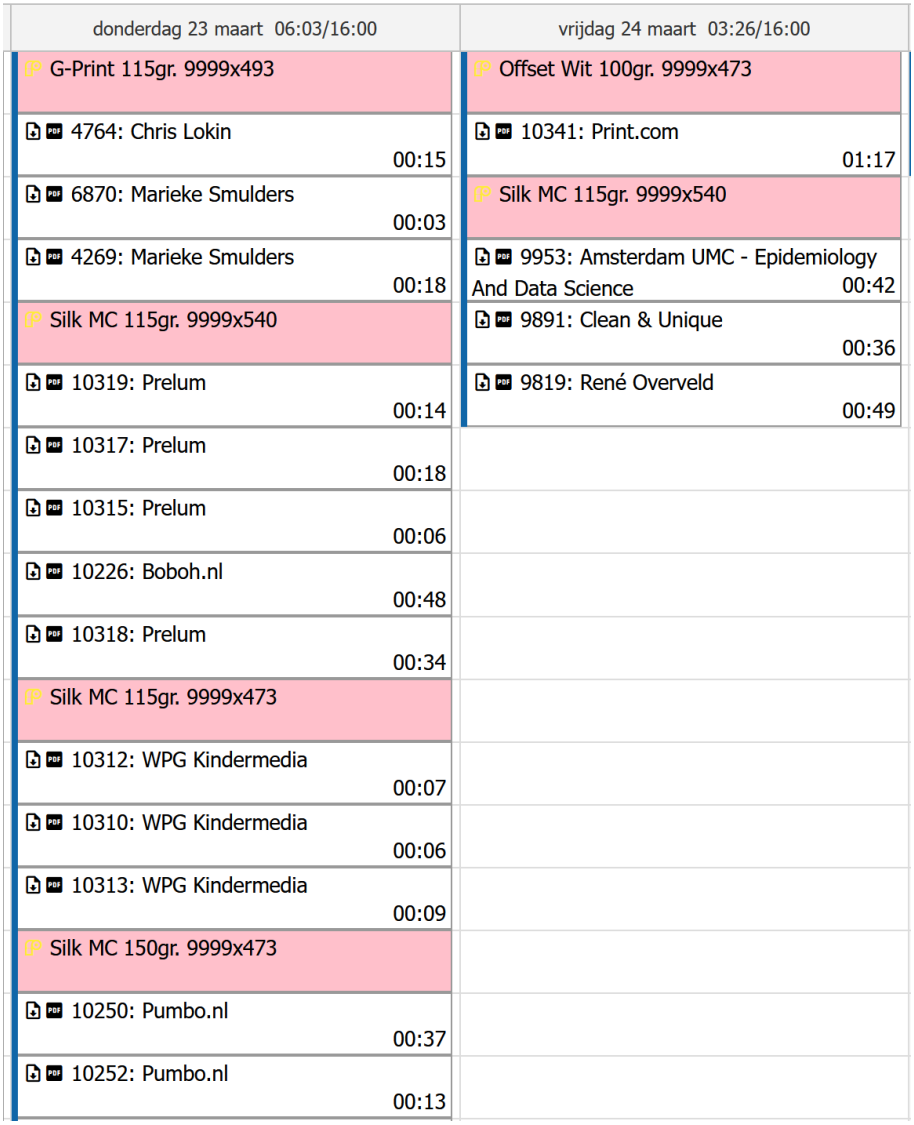

Figure 9.1: Planning view per day with materials

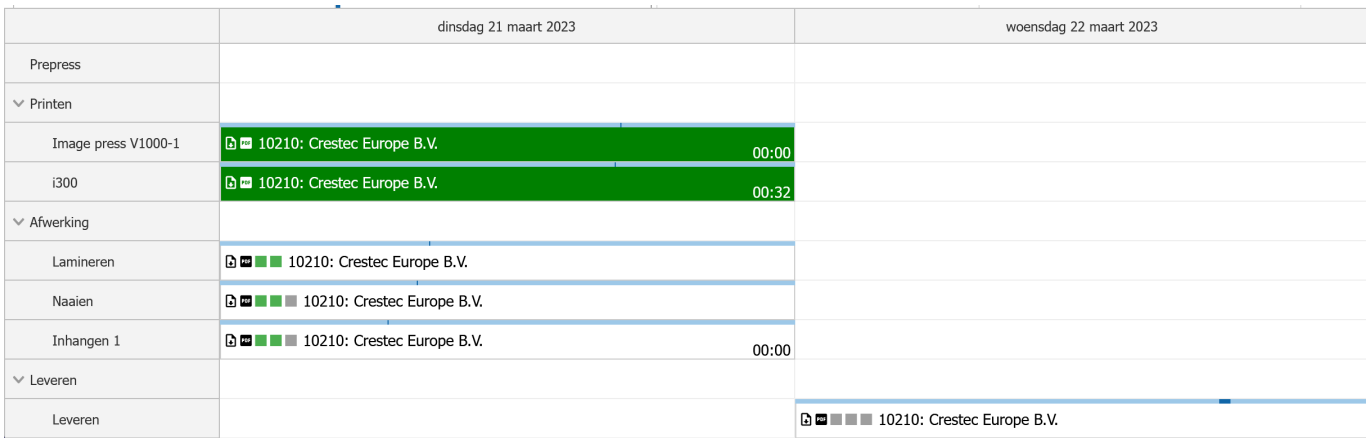

Figure 9.2: Planning per order with finished components

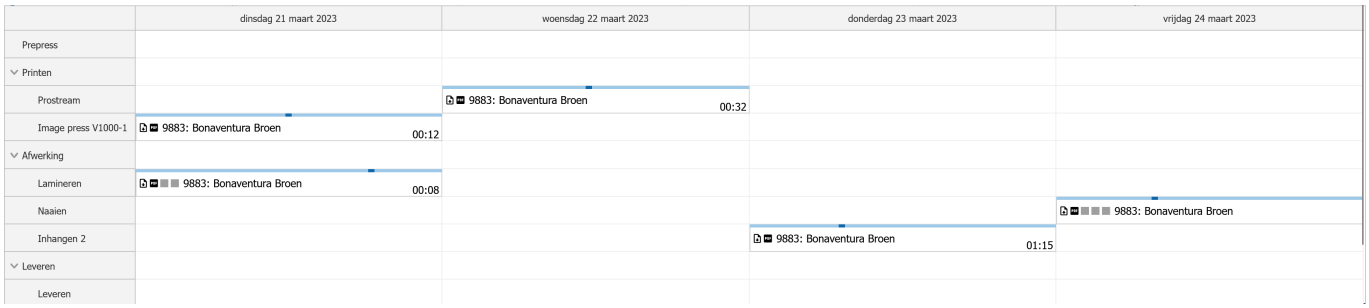

Figure 9.3: Second planning view per order

Bibliography

Brusco, M. J., Thompson, G. M. & Jacobs, L. W. (1997). A morph-based simulated annealing heuristic for a modified bin-packing problem. *Journal of the Operational Research Society*, *48*(4), 433–439. https://doi.org/10.1057/palgrave.jors.2600356

Chou, F.-D., Wang, H.-M. & Chang, P.-C. (2008). A simulated annealing approach with probability matrix for semiconductor dynamic scheduling problem. *Expert Systems with Applications*, *35*(4), 1889–1898. https://doi.org/10.1016/j.eswa.2007.08.100

Collinot, A., Le Pape, C. & Pinoteau, G. (1988). Sonia: A knowledge-based scheduling system. *Artificial Intelligence in Engineering*, *3*(2), 86–94. https://doi.org/10.1016/0954-1810(88)90024-6

Conway, R. W., Maxwell, W. L. & Miller, L. W. (1971). Theory of scheduling, 1967. *Addison-Wesley, Reading, Mass.[: 5] M. EISENBERG, TwO queues with changeover times, Operations Res.(2)*, *19*, 386–401.

Dearing, P. & Henderson, R. (1984). Assigning looms in a textile weaving operation with changeover limitations. *PRODUCT. INVENT. MANAGE.*, *25*(3), 23–31.

Fabrycky, W. & Shamblin, J. (1966). A probability based sequencing algorithm. *Journal of Industrial Engineering*, *17*(6), 308.

França, P. M., Gendreau, M., Laporte, G. & Müller, F. M. (1996). A tabu search heuristic for the multiprocessor scheduling problem with sequence dependent setup times. *International Journal of Production Economics*, *43*(2-3), 79–89.

Gupta, J. N. & Darrow, W. P. (1986). The two-machine sequence dependent flowshop scheduling problem. *European Journal of Operational Research*, *24*(3), 439–446.

Hakimzadeh Abyaneh, S. & Zandieh, M. (2011). Bi-objective hybrid flow shop scheduling with sequence-dependent setup times and limited buffers. *The International Journal of Advanced Manufacturing Technology*, *58*(1-4), 309–325. https://doi.org/10.1007/s00170-011-3368-5

Jin, Z., Yang, Z. & Ito, T. (2006). Metaheuristic algorithms for the multistage hybrid flowshop scheduling problem. *International Journal of Production Economics*, *100*(2), 322–334.

Johnson, S. M. (1954). Optimal two-and three-stage production schedules with setup times included. *Naval research logistics quarterly*, *1*(1), 61–68.

Katragjini, K., Vallada, E. & Ruiz, R. (2013). Flow shop rescheduling under different types of disruption. *International Journal of Production Research*, *51*(3), 780–797.

Kirkpatrick, S., Gelatt Jr, C. D. & Vecchi, M. P. (1983). Optimization by simulated annealing. *science*, *220*(4598), 671–680.

Lawrence, S. R. & Sewell, E. C. (1997). Heuristic, optimal, static, and dynamic schedules when processing times are uncertain. *Journal of Operations Management*, *15*(1), 71–82.

Masson, R., Vidal, T., Michallet, J., Penna, P. H., Petrucci, V., Subramanian, A. & Dubedout, H. (2013). An iterated local search heuristic for multi-capacity bin packing and machine reassignment problems. *Expert Systems with Applications*, *40*(13), 5266–5275. https://doi.org/10.1016/j.eswa.2013.03.037

Matsuura, H., Tsubone, H. & Kanezashi, M. (1993). Sequencing, dispatching and switching in a dynamic manufacturing environment. *The International Journal of Production Research*, *31*(7), 1671–1688.

Metropolis, N., Rosenbluth, A., Rosenbluth, M., Teller, A. & Teller, E. (1953). Simulated annealing. *Journal of Chemical Physics*, *21*(161-162), 1087–1092.

Mirsanei, H. S., Zandieh, M., Moayed, M. J. & Khabbazi, M. R. (2010). A simulated annealing algorithm approach to hybrid flow shop scheduling with sequence-dependent setup times. *Journal of Intelligent Manufacturing*, *22*(6), 965–978. https://doi.org/10.1007/s10845-009-0373-8

Naderi, B., Zandieh, M., Khaleghi Ghoshe Balagh, A. & Roshanaei, V. (2009). An improved simulated annealing for hybrid flowshops with sequence-dependent setup and transportation times to minimize total completion time and total tardiness. *Expert Systems with Applications*, *36*(6), 9625–9633. https://doi.org/https://doi.org/10.1016/j.eswa.2008.09.063

Naderi, B., Zandieh, M. & Roshanaei, V. (2008). Scheduling hybrid flowshops with sequence dependent setup times to minimize makespan and maximum tardiness. *The International Journal of Advanced Manufacturing Technology*, *41*(11-12), 1186–1198. https://doi.org/10.1007/s00170-008-1569-3

Nawaz, M., Enscore Jr, E. E. & Ham, I. (1983). A heuristic algorithm for the m-machine, n-job flow-shop sequencing problem. *Omega*, *11*(1), 91–95.

Pinedo, M. & Hadavi, K. (1992). Scheduling: Theory, algorithms and systems development. *Operations research proceedings 1991* (pp. 35–42). Springer.

Polyakovskiy, S. & M'Hallah, R. (2021). Just-in-time two-dimensional bin packing. *Omega*, *102*, 102311. https://doi.org/https://doi.org/10.1016/j.omega.2020.102311

Radhakrishnan, S. & Ventura, J. A. (2000). Simulated annealing for parallel machine scheduling with earliness-tardiness penalties and sequence-dependent set-up times. *International Journal of Production Research*, *38*(10), 2233–2252. https://doi.org/10.1080/00207540050028070

Romeo, F. & Sangiovanni-Vincentelli, A. (1991). A theoretical framework for simulated annealing. *Algorithmica*, *6*(1), 302–345.

Ruiz, R. & Stützle, T. (2008). An iterated greedy heuristic for the sequence dependent setup times flowshop problem with makespan and weighted tardiness objectives. *European Journal of Operational Research*, *187*(3), 1143–1159.

Salvador, M. S. (1973). A solution to a special class of flow shop scheduling problems. *Lecture Notes in Economics and Mathematical Systems*, 83–91. https://doi.org/10.1007/978-3-642-80784-8_7

Schaller, J. E., Gupta, J. N. & Vakharia, A. J. (2000). Scheduling a flowline manufacturing cell with sequence dependent family setup times. *European Journal of Operational Research*, *125*(2), 324–339. https://doi.org/10.1016/s0377-2217(99)00387-2

Van Laarhoven, P. J., Aarts, E. H. & Lenstra, J. K. (1992). Job shop scheduling by simulated annealing. *Operations research*, *40*(1), 113–125.

Wang, K. & Choi, S. (2012). A decomposition-based approach to flexible flow shop scheduling under machine breakdown. *International Journal of Production Research*, *50*(1), 215–234.

Zegordi, S., Itoh, K. & Enkawa, T. (1995). A knowledgeable simulated annealing scheme for the early/tardy flow shop scheduling problem. *The International Journal of Production Research*, *33*(5), 1449–1466.