31/Jan/2022

The Industrial Bakery Scheduling Problem

Solving a complex Continuous Hybrid Flow Shop model using meta-heuristics

Project Report

Bosch, Yorick
UNIVERSITY OF TWENTE – INDUSTRIAL ENGINEERING & MANAGEMENT
2022

Management Summary:

Can a company improve the performance of their existing machine lines with some simple scheduling changes? Royal Kaak Group (colloquially: Kaak) is a company that designs, manufactures and maintains machines and complete machine lines for professional industrial bakeries. In one such line, the Kaak small-bread line, changing customer requirements create many challenges for Kaak to overcome. One of these challenges is when customers want to bake many different types of bread on their line in one production run: whilst the individual machines that make up the lines are built and calibrated to function in near-perfect unity when only considering a single bread recipe, they require time for change-over and cleaning when switching to and from different recipes. These changeover times vary per machine, and per combination of bread recipes, which causes big scheduling problems. Some lines can process over 30+ different recipes.

Since customers may decide to create a production schedule for any combination of these recipes, in any batch size, this problem grows out of control fast. Imagine: two recipes only have two ways in which they can be arranged, but four recipes already have 4*3*2*1 = 24 different arrangements. A production plan of ten recipes in a day has ~3.6 million different arrangements. Now add variables like batch sizes and machine variants into the mix, and you have a problem that is essentially infinite in size. Without a proper tool, reliably finding the best sequence for production plans is impossible. We call this: the Industrial Bakery Scheduling Problem.

In addition to the problem of formulating the best production plans, Kaak has an additional problem in finding out which machine upgrades would most efficiently cut down on change-over times. New or improved variants of many machines in the line generally provide the same level of continuous production, but cut down on change-over times and other types of unproductive waiting time. However, to see which machines provide the best outcomes for both Kaak and its customers, and which are thus worthy of either the investment in initial R&D or purchase, a tool is needed to compare the time savings of all machines depending on the target production plan.

Finding an optimal production schedule becomes a major challenge for Kaak and its' customers in these cases. Inefficient scheduling can lead to hours of lost production time per day, costing bakeries extra wages, energy costs, and generally making their business less competitive. This should be more than enough reason to tackle the problem. In this thesis we attempt to solve both the scheduling- and machine-variant problems in a two-part solution.

First, we built a custom flow-shop model of the small-bread line.

- The model is capable of near-instantly calculating the makespan of any combination and sequence of bread recipes, in any combination of batch sizes.
- The model takes into account recipe-specific production times, sequence-dependent change-over times and any combination of batch sizes.
- The model is capable of doing the above on any of 16 distinct versions of the machine line, representing four machines that have a significant alternate variant.
- The model calculates waiting times and suggests a precise "postponement" time for each recipe to avoid said waiting times, and ensure continuous production after a batch enters the line.
- The model includes all machines from the dough-dispenser to the (optional) freezer.
- We combine several of the smaller, less time-consuming production steps into a single cohesive step to help save on calculation time without affecting model outcomes.
- In theory the same model could be applied to other continuous machine lines

Secondly, we used the outputs of this model to optimize the makespan of different production plans and machine arrangements.

- By switching or inserting recipes, the sequence of recipes in a production plan is altered.
- We use both construction and improvement heuristics to help explore the huge solution space within a reasonable timeframe, and find better results.
- This leads to quick, (near-)optimal solutions to the industrial bakery scheduling problem, and helps us find the best setup of machines to handle different types of production plans.

For validation, we solved an example case concerning the best production sequence for a line where extra recipes were added. We also ran two sets of experiments on 16 different machine line variants, with production plans of both few and many recipes, as well as small and large batches. From this case and these experiments we can deduce the following results:

- We found clear solutions to the scheduling problem in the example case, answering all questions.
- We found direct- and interaction effects of the novel machine variants on the average makespan for a wide variety of production plans of different lengths and size, which gives Kaak new insights into their own lines that were hard to quantify before.
    - In all cases, adding a freezer module will lead to increases in average makespan.
    - For production plans with multiple small batches, installing a double cooling spiral can significantly cut down on makespan (-20%) especially if a freezer module is enabled.
    - In almost any production plan, significant time savings can be achieved with the multi-tower rising chamber model, compared to the normal variant (-10%).
- Regardless of machine variants, the average savings of the best found production plan can represent hours of real-life time in a running factory per day.
- In an average production plan with change-over times, the effect of advanced machine variants is a decrease in change-over times and an increase in average productivity per time unit.

This tool will help Kaak provide instant value to its existing customers, by significantly reducing lost productivity in existing lines, or allowing for more flexibility in baking different recipes on the same production day. It can also help Kaak to make more informed decisions about potential future investments in new machine types, as well as help them convince customers of the added value of these machines. Calculating a set of (near-)optimal solutions for a single production plan only takes a few minutes and could be integrated into existing work-flows for engineers and salespeople. Since this work did not happen before in any efficient, centralized manner, it represents an enormous improvement that should save many hours of misplaced engineering time.

The model provides a solid basis on which to recommend production plans, or to help engineers to better design new lines. We recommend further data gathering regarding processing- and change-over times of different bread types, to make the model more precise and to enable Kaak to simulate more production steps such as the mixing- and pre-rise times. The model could also be applied to other machine lines, such as the Kaak pizza-pie line and loaf-bread lines, or other continuous production lines in general if they are similar in nature.

In our model we did not take stochasticity into account due to lack of data. Using the outcomes of the model as a basis for further research into Discrete Event Simulation (DES) is encouraged: with a good estimation of efficient production plans, DES could help to further optimize machine lines and to spot inefficiencies or design errors in the line at a high logistical level, before the lines are further designed, built and installed at customers. For the set of discrete problems that are hard to simulate in flow-shop models, we recommend the use of DES in addition to the outcomes of our flow-shop model.

Table of contents

Chapter 1: Introduction and Research plan

1.1) Company information

Royal Kaak Group is a producer of stand-alone machines and complete production lines for a wide range of mass produced bread products. They are located in the eastern Netherlands, with various branches, subdivisions, sales points and customers across the globe. They employ around 800 people, ranging from engineers to factory employees and sales personnel. They pride themselves on their motto: From Silo to Truck, pointing to the all-encompassing nature of their service and production lines. Royal Kaak aims to provide their customers in the food industry, such as producers of bread and pre-fab pizza pies, with the technical solutions they need to allow them to make the industrial bread products they desire. For brevity, we will refer to the entire company as "Kaak" from now on.

1.2) Research motivation

Kaak produces, sells and offers support for a wide variety of stand-alone machines and machines that are a part of a complete production line. The production process of any type of bread can be near-continuous, but there are still many differences between bread types, from artisanal-style baguettes to mass-produced pizza pies. In their most advanced machine lines all the machines are linked together with conveyor belts to eliminate human intervention in the process. In order to help their customers make the difficult choice of picking the correct machine for their desired outcome, Kaak makes use of simulation. Using conventional, flow-based estimations they are usually able to gauge the effectiveness of different setups. However this does not hold true for all machines and all situations.

One problem Kaak regularly encounters has to do with their small-bread line, a near-continuous machine line that can be used to produce a wide variety of small bread types[1], which we will refer to as "recipes". This line is physically quite large and includes all steps in bread making, from the mixing of the dough to the eventual packaging of finished products. Whilst the small-bread line is designed and equipped to handle continuous production of any the small-bread recipes, many bakeries employ the line to bake multiple batches of bread per day. Since these recipes are all individually unique, this places extra demands on the line compared to continuous single-type production. Each change in recipe can mean a new setting for each individual machine, and causes change-over time: for example because the temperature of the oven needs to be changed, or due to cleaning to ensure cross-contamination between two recipes stays minimal. However, these change-over times are independent per machine, and multiple recipes can be on the line at the same time.

1.3) Problem statement

Customers may come to Kaak with a question about, for example:

- How to get the maximum amount of production out of their existing machine line?
- Which sequence of recipes to pick for a particular production order?
- Which machines to upgrade or add to their line to reach a new production target?

These are questions which the engineers at Kaak then have to answer within a reasonable time frame, with great accuracy and at the lowest possible cost to their clients. Since there are many possible combinations of machines of different capacity, since each recipe has different production

[1] As opposed to big loaves of bread. Think of ciabattas, baguettes, pumpkin-seed buns, etc.

times per machine and since machines experience a varying amount of change-over time between each recipe, this is not an easy problem to solve. A production sequence that may be optimal for a certain machine may result in huge waiting times for a machine later down the line. There are more than 30 types of bread that can be baked on the same line. Since customers may come up with any combination of recipes they want to bake on a certain day, this presents a clear central managerial problem: what is the best sequence to bake any random set of bread recipes on the small-bread line?

However, this is not enough for some of the engineers at Kaak, who are also struggling to find a correct answer to questions about which machines to recommend to their customers. Often these types of problems are solved by using an "overkill" method, by building or recommending machines that offer more functionality than is actually required for the job. Sometimes the customer has to alter their bread recipes, to ensure the bread fits the line instead of the other way around. The baking industry is very competitive and therefor Kaak wants to change this practice. Selling equipment to their customers that is too powerful or unfitting for the job makes them less competitive. A central directive must be: sell the customer the lightest machines that are capable of doing the specified job and ensure the customers have the information they need to pick the most optimal production schedule for their desired production targets.

The engineers at Kaak are currently ill-equipped to answer these types of problems. Not only is the problem multi-faceted, it is also combinatorically huge: the composition of the production line can vary from factory to factory (possibly with machines from Kaak's competitors mixed into the line), and different compositions will perform better with certain bread types. Then also, for each bread type, the machines will each have their own production and change-over times, depending on the sequence of bread types and their production quantity. Finally, there is a near-infinite amount of different sequences for the production plans. It is clear that Kaak needs a new way to simulate the small-bread line, to enable their staff to provide quick answers to their customers' inquiries.

Kaak already experienced problems with configuring the complete automated machine line. Currently their tools are focussed on providing optimal movement and performance for the individual machine models, but no comprehensive tool is in use to compare the functioning of all machine models in a line. Kaak wants to own a tool they can use to quickly plug in potential machine arrangements to simulate the throughput metrics. This tool should eventually enable them to build a complete machine line from a library of machines and conveyor belts at the touch of a button. They initially reached out to the University of Twente to help them set up a preliminary Discrete Event Simulation (DES) tool of the small-bread production line, to see if it would be a good fit for this purpose, as well as for research into personnel movements and PLC programming issues. However DES is not an efficient solution method for huge combinatorial problems like the one above: whilst by no means slow, it still takes a while to run. DES is better suited to testing known inputs and finding the system specific performance metrics. Optimization of these inputs is better done separately and it seems this is the bigger problem at play here. Solving this "Industrial Bakery Scheduling problem" will be a first step to improving the other challenges that Kaak faces.

1.4) Research goal

The goal of this thesis is to create an efficient method to solve multiple problems at once: both to provide a good estimate of the performance of a production plan on the small-bread line, but also to find the most efficient sequence of recipes. When a preferable combination of machines and production schedules has been found, these settings can then eventually be imported into a more

advanced simulation tool like DES to properly predict the actual throughput and bottleneck statistics, which can be relayed back to the customer. Alternatively these settings could be directly relayed to the Kaak engineering or sales team, to help them with their ongoing work.

The deliverables of the study will be two-fold: firstly, a deterministic flow-shop model that will provide a quick and efficient way to find the makespan of any given production plan on multiple machine arrangements. Secondly, a set of optimization methods to help provide optimal or near-optimal solutions to the type of combinatorial problems experienced by Kaak. This should result in a robust set of solutions, performance metrics and answers to many of the questions that customers can ask of the engineering department. That will form the basis of answering the problems faced by Kaak, improving their customer service and overall competitiveness.

1.5) Problem approach

We will construct a deterministic optimization model in the form of a custom flow-shop model that can be used to vastly narrow the search area for good solutions, depending on the input given by the user. This model will be structured around the use of a goal function, which will calculate the makespan value of a given set of machine settings and production plans. Then, through the use of a constructive heuristic and meta-heuristics, a selection of the best and/or most cost-efficient solutions can be gathered. Flow-shop modelling is a common practice in Operations Research and it should offer a good solution for this problem, although we will have to expand upon basic versions to make the problem fit.

Data about the functions, variations and performance of the existing machines in the line will have to be gathered, be it through measurements, interviews with engineers, looking at raw designs, or readily available performance data gathered from existing customers. Working on an example customer support case will help us to validate our model further, seeing if the model conforms to reality, and if new bread types can quickly be introduced and optimized. We can then test the performance in different situations, with different recipes. Finally, this should help us answer the optimization questions posed by Kaak.

1.6) Research questions

We shall investigate the following main questions, and related sub questions in the order as depicted below, with each number representing a chapter:

Chapter 2. Analysis of current situation

The first step of the study will be to research and understand the complete small-bread machine line. Also, an overview needs to be made of the current simulation techniques being employed by Kaak, to see if and where they fall short. The following sub-questions will be addressed:

1. What processes occur in the small-bread line?
    I. What machines are present, in what sequence, what are their metrics?
    II. What variations of the machines in the line exist?
    III. What bread recipes are made on the line, what are their characteristics?
    IV. What is the state of simulation and optimization software/processes within Kaak?
2. What kind of information does Kaak require to be able to optimize their systems?

Chapter 3. Literature review

In order to establish a good understanding of the simulation methods we are going to employ, we will perform a literature review on the flow-shop problem based on the following questions.

1. What solution methods are usually employed to solve flow-shop scheduling problems?
    I. How do these solution methods apply to the Kaak case?
    II. What alterations need to be made to a standard problem to make it fit?

Chapter 4. Developing the flow shop model

1. How will we compute the makespan of a solution?
    I. How do we deal with the near-continuous, batch-driven nature of the production process?
2. What parts of the full machine line must be represented in the flow-shop Model?
    I. Do all machines have an equal impact on the value of the solution?
    II. Which, if any, machines can be combined into more convenient blocks?
    III. What is the impact of advanced machine variants on our model?

Chapter 5. Developing the optimization schemes

1. Based on the flow-shop model that was developed, and the literature review, which optimization techniques do we employ and how?

Chapter 6. Validation and Experimentation

While validating the model and experimenting with the results, we will answer the following questions:

1. What are the outcomes of our model? Do these conform to reality?
    I. Can a real customer question be answered?
2. What is the value of this model?
    I. Does our optimization method lead to improved performance or lower costs?
    II. What conclusions can we make about the effectiveness of our algorithms?
    III. Which machines would make for good additions in which circumstances?

Chapter 7. Conclusion and Discussion

Finally, our conclusion and discussion of the results will revolve around the following questions:

1. Is our model a sufficient solution to the optimization problems encountered by Kaak?
2. What avenues of further research should be pursued?

Chapter 2: Analysis of the current situation

In this chapter we will take a broad look at the current situation at Kaak, to delineate the machines and processes that happen in the small-bread line, in as far as they are relevant to our research. First, we discuss the way we measure the performance of the line, then the various machines that make up the small-bread line and the way they interact, before giving an overview of the characteristics of the bread types we include in the model and the current state of simulation at Kaak. Finally we will discuss similar problems that occur elsewhere in the food industry.

2.1) Scope

For the purposes of our study, baking any type of bread starts with the dispensing of chunks of dough, and ends with the packaging of a finished bread product. We will not include the delivery and storage of raw materials into a production facility, nor will we discuss the storage and delivery methods of the finished bread products. These factors change considerably from facility to facility, and are outside the influence of Kaak. It would therefore be a reasonable assumption that the only parts of the line affected by our planning are those that are directly built and delivered by Kaak. We also do not model employee behaviour during production hours. We will refrain from going into deep, technical details of the different machine models, but stick to overall function and the ways that we can control the bread-making process from a day-to-day perspective. In that regard, our focus is on the tactical level.

2.2) Batch sizes and processing times

On the small-bread line processing times for each bread recipe are fixed, regardless of batch size. If a single bun has to bake in the oven for 15 minutes at 200 °C, then a thousand buns also have to bake for 15 minutes at 200 °C. However, batches of bread move across the line in a continuous stream. Whilst the first bun in a batch may be half-way through the machine line, the last bun may still be in the mixing bowl. Thus for each bun in a batch the same processing time will apply across all machines. However, these processing times will start and end at different times. Batches do not behave like may be expected in normal models, where an entire batch is loaded into a machine before being processed. Rather, the batches enter and exit the machines as if they were on a continuous conveyor belt, without interference. This means that the first bun in a batch is the first to enter and to leave a machine. Depending on the size of the batch, the last bun may then take seconds or hours to enter and leave the same machine. The rest of the buns in a batch will necessarily fall in between the first and last. If no breakdowns occur, which we will assume for the sake of simplicity in the model, each batch will behave in a very predictable manner.

Change-over time is a big factor in the small-bread line. We define it as the time it takes to change the settings or tooling, and perform any required cleaning or inspection on a machine, to ensure it is ready to handle the next recipe in a production plan. Waiting time could occur when one recipe has not exited a machine where another is due to enter, when there is insufficient space between two recipes to allow for the required change-over time. Waiting time could imply shutting down the entire line up to the point where the waiting is required, since batches of bread move along the line continuously. This could have consequences for unbaked bread that is still rising, bread that is stuck in a hot oven, bread that gets continuously covered in an ever-increasing pile of sprinkles, etc. To ensure the quality of bread that comes out of the factory, no scheduled waiting time is allowed on the line. Waiting time is therefore not allowed, or at least highly discouraged, in real-life factories. It might still be unavoidable due to breakdowns, but we will not consider it as a standard. Instead Kaak plans a so-called “postponement time”, which is essentially waiting time at the start of the production process, to eliminate on-line waiting times and ensure each recipe can keep moving

continuously after it enters the production line. Even though it is necessary, this postponement time is currently not optimized.

We measure the performance of the line in terms of total makespan. We define makespan as the time between first entry of the first recipe in a production plan onto the first machine in the line, and the time the final bun of the last recipe exits the last machine on the line. Since the processing times are known and change-over times depend on the sequence of recipes in a production plan, finding a production plan that will produce all recipes at the appropriate batch size within the shortest amount of time is a realistic, often-stated goal by production managers.

For at least a part of each machine-line, the buns are carried across the line on product carriers. On the small-bread line so-called “peel boards” (PLBs) are most common. They are essentially flat trays that help keep the buns in shape and in position. See Figure 1 for an example of PLBs carrying unbaked buns on the production line. We will purely focus on the case of PLBs in the small-bread line, although different types of product carriers do exist, and impact the naming of several machines.

One peel board may hold anywhere from 12 to 100+ individual buns of bread, depending on the size and shape of the bun. Since customers may come up with any combination of bread recipes and batch sizes, theoretically from 1 to 1000+ PLBs per batch, our model will have to be very flexible in allowing these combinations and handling them. Each recipe has an associated tact time, generally around 8 seconds, which is the rate at which PLBs move around the system. So after some start-up time, the average rate of production of any recipe is around 1 PLB/8s. Since it is computationally much more convenient to discuss the processing time of a peel board in this system, we will take the peel board as our unit of bread measurement. Note that the PLBs are separated from the buns before they enter the oven, and that afterwards the calculations refer to “one peel board worth of buns”. The process of storing and cleaning the PLBs will be kept out of the equation as again, this is not experienced to be a bottleneck. Adding extra PLBs to a system is a matter of literally stacking them in a storage area, where an automated robot will handle the rest.

Figure 1: Peel boards carrying products on the production line along a conveyor belt (from Kaak website)

2.3) Machines and variations

Listed in Table 1 are the different machines in the small-bread line, in order of appearance in the factory. It can be assumed that any machine is connected to its' predecessors by automatic conveyor belts. The time indication is the time it would take a peel board worth of dough or bread to pass through this step. Important to realise here is that this timespan occurs for each individual bun, but that all buns are part of a batch that undergo the process continuously as noted in Section 2.2. As such, when 80 PLBs worth of buns move through the forming station, the total processing time for the batch, or the time it occupies at least part of machine, is not a multiple of $30\ sec\ *\ 80\ PLBs$, but rather $30\ sec\ +\ (80\ PLBs\ *\ \sim 8\ sec\ tact\ time)$.

The change-over time is the maximum time that may be required between two opposing recipes: the minimum is always 0, although intermediate values are possible. All these machines will have to be represented in our model to make it accurate, although for the purposes of modelling many can be combined into a single step because of their short duration and relatively low and stable change-over times. For example, the dough dispenser and forming station will always be turned on or off together because they are interdependent, and as such we can combine their production- and change-over times. Many machines have alternate variants that do not significantly impact production times. For example, the sprinkler system can be altered for many different kind of toppings, whilst performing identically. Only four machines have a significant alternate variant that is relevant to the makespan. These are the rising chamber(s), oven, cooling spiral and freezing step. Their differences are briefly explained in the table.

Table 1: Machines of the small-bread Machine line

Machine/ Step	Description	Production Time/PLB	Max Change-Over Time
Dough dispenser	The dough is poured out of a mixing bowl into a funnel, leading to a machine which will cut the dough into chunks. There are several variations of chunks, depending on the recipe. One variant creates the individual chunks that become buns, another creates a series of large chunks that are rolled into a slab.	8 seconds	10 minutes
Forming	Chunks are rolled, cut or stamped into even-sized doughballs in several steps on a conveyor line. Each recipe has a machine pattern, although most use similar tooling with small variations.	30 seconds	3 minutes
Panning-point	The individual dough balls are carefully and precisely laid in a pre-determined pattern on a peel board. One machine can handle most combinations. In other lines this machine puts dough-balls in their "pans", a different type of product carrier, hence the name.	8 seconds	2 minutes
Rising chamber(s) /Climas	The peel boards enter the rising chamber, a tower where temperature and humidity are kept at a steady level for optimal rising conditions. The peel boards move slowly up and down the tower, before exiting out the other side. A more advanced version of this tower consists of multiple towers, that can be partially or completely skipped. This machine, or set of machines, is often referred to as "Climas", for "Climate Chamber".	60-120 minutes	1 minute
Turning	Depending on their orientation, the peel boards are turned 90°	8 seconds	1 minute
Sprinkler	Depending on the recipe, the bread is decorated with a sprinkle of flour, moon seed, pumpkin seeds, etc. Some machine lines have multiple sprinklers with different contents, others simply swap a container between recipes.	8-36 seconds	6 minutes
3-Carrier collector	Three peel boards are collected and placed side-by-side, to take advantage of the width of the oven. Can also be used for other product carriers without problem, for example baking loafs of bread in pans.	24 seconds	1 minute
Cutting Robots	Knife-wielding robots place precise cuts across the top of the buns, or let the buns pass by uncut, depending on the recipe.	24 seconds	5 minutes
Scrabbler	The buns are lifted off of their peel boards and placed on the oven conveyor belt in several steps, with space allocated in between them. The peel boards are separated and sent to a different line where they are cleaned, stored and re-used.	24 seconds	1 minute
Oven	A long horizontal oven, with a belt that can change speed depending on the bread type. Some ovens have the ability to heat sections independently, which reduces total change-over time from bun to bun. The length of the oven changes from factory to factory, long ovens having more capacity.	10-20 minutes	25 minutes
Fakir	This machine gathers batches of buns as they leave the oven, and aligns them with the next set of conveyor belts. The name comes from the pins that are sometimes used to relieve baked bread from their pans: just like the mythical fakirs of the orient and their bed of nails.	24 seconds	1 minute
Cooling Spiral	A long spiralling tower where the individual buns can cool when they exit the oven. Cooling times can differ wildly depending on the size and temperature of the bun. Variants exist with multiple separate cooling towers for multiple batches cooling at different speeds. Often the cooling length of buns is altered to fit the tower, instead of the other way around.	1-2 hours	1 minute
Freezing Spiral	Exact copy of the cooling tower, but colder. Not all bread is frozen, so an alternate variant of this line is the same, but without the freezer. Note that change-over time is 0, as it always follows the cooling tower.	1-2 hours	0 minutes
Packaging	Different bread types take different packaging methods, but most are packaged in plastic bags. Machines are prone to jamming. However, often delivered by outside contractors, hard to get solid data.	8 seconds	1 minute

In Figure 2, the different steps in the production process are laid out in minutes. This graph puts the vast differences in average processing time into perspective: many of the steps hardly even show up in the graph. However, all steps have a measure of change-over time associated with them which we must take into account.

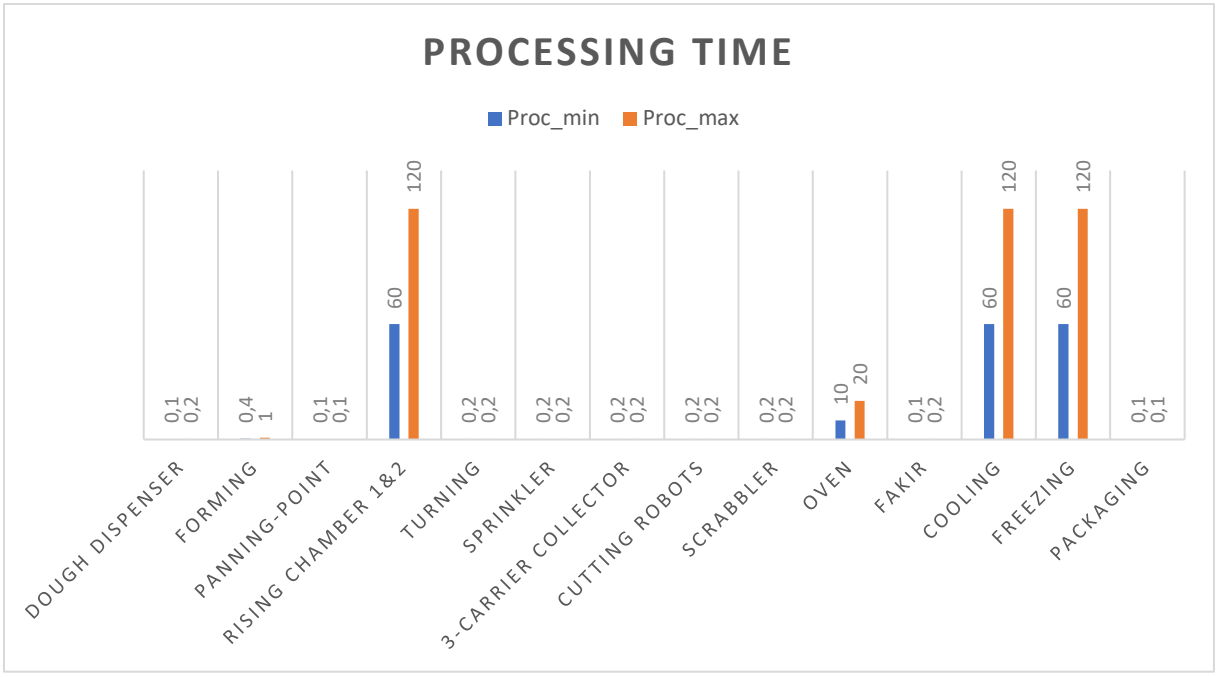

Figure 2: Minimum and maximum processing time per peel board per production step in minutes per peel board

We omit one phase from our data, and that is the so-called “pre-rise” phase. This includes the time in which ingredients are first mixed, before the dough is dispensed. We were unable to collect enough accurate data on this step in the production process. There may be considerable differences to be found in some factories concerning the pre-rise phase. Some omit it altogether and dispense the dough immediately after mixing. Others let their doughs rest for up to 6 hours before dispensing, to help gluten development.

In some bakeries it is common to start the bulk fermenting process for a “slow” batch with a long resting time first, before starting the mixing of a different “fast” type of bread. This fast batch can then enter the production line before the first. That is the only known instance of recipes switching production sequences. Apart from that, recipes cannot switch production sequence: on the part of the line connected by conveyor belts a batch of bread needs to complete a step before the next batch is allowed to enter. A visual overview of the machines in the line can be seen in Figure 3.

It should be noted that the carrier handling system is also not a part of our flow-shop or optimization procedure. The carriers are all the same in this version of the model, and their handling does not appear to be a bottleneck.

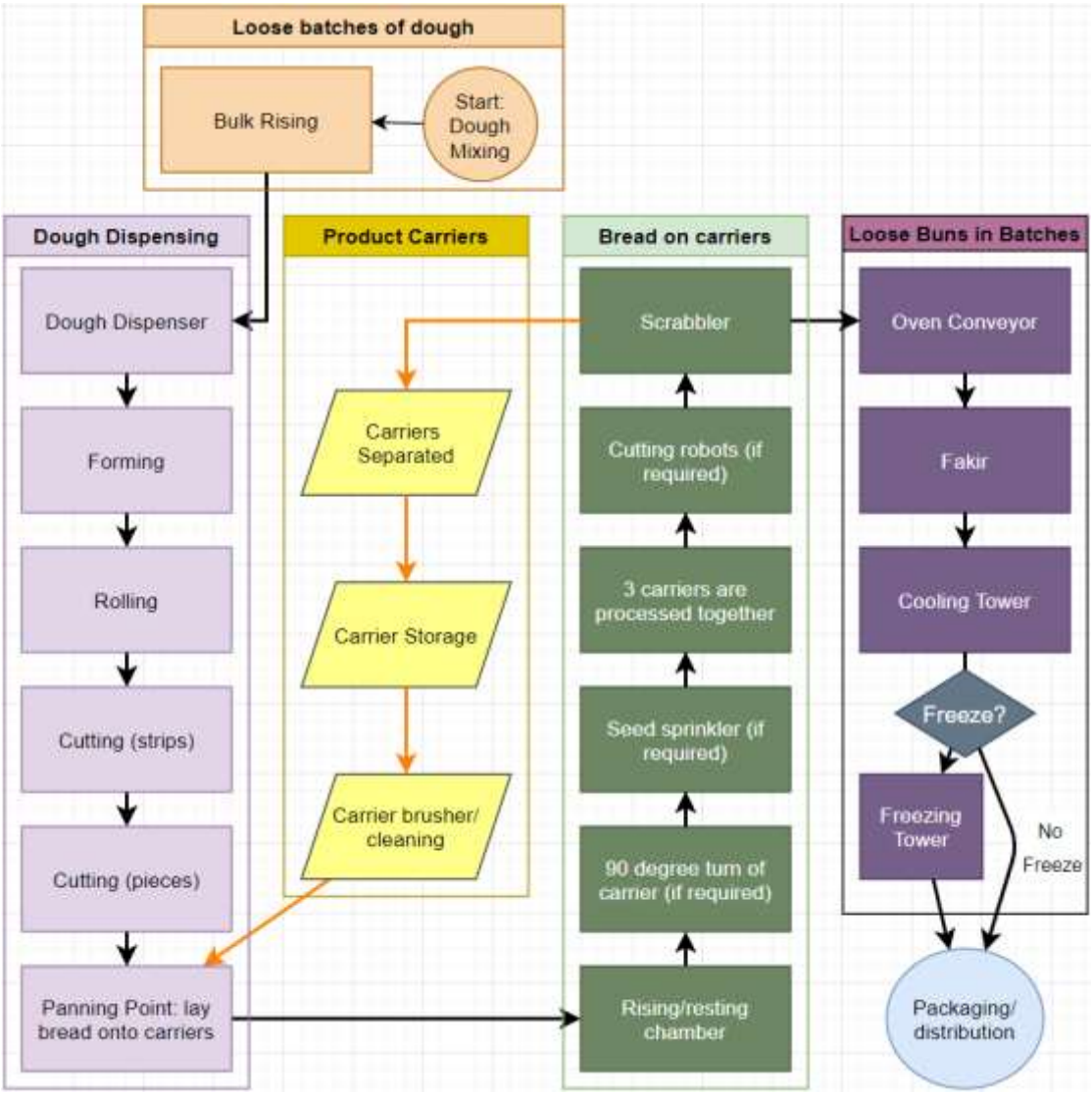

Figure 3: Small-bread line Flow Chart, includes all steps. Note that the "Loose batches of dough" and "Product Carriers" sections are outside of scope.

2.4) Recipes

Listed in Table 2 below are the characteristics and production data that should be gathered for each bread type. This includes the processing time for each of the machines listed in Table 1. It also includes the product-specific characteristics like the baking temperature and required decoration pattern, and the average production rate or Tact Time of this bread type (in seconds per PLB).

Table 2: Bread Characteristics

Name	**Value**	**Notes**
Recipe Number	r : Integer	
Recipe Name	String	
Processing time	*P(m,r)*	In minutes, per Machine *m*, per Recipe *r*
Tact Time	Seconds	Per peel board
Dough type	White/Brown	Could be extended for rye, etc.
Turner	True/False	Does the Peel Board need to be turned 90°?
Decoration	True/False	Every 'true' decoration needs change-over in sprinkle station
Cuts	True/False	Every 'true' cut needs different settings of cutting robots.
Baking Temperature	°C	Change-over depends on heating/cooling potential of the oven.

At one bakery where a small-bread line is used, at least 21 different recipes can be produced on the same line. This bakery will rarely produce more than ~12 recipes in a day, but their line has the ability to produce all of them. Their list of recipes is by no means an exhaustive list of all types of bread that could be produced, but it is a good starting point for our data collection. It contains both large and small bread types, of various dough types, decoration and baking settings. If a new bread type would need to be introduced to the model, it could be modelled by adding it to the list with its characteristics, after which it would function just like the rest.

Note that Table 2 does not list change-over times; these are not set in stone and change depending on the sequence of recipes in the production plan. The product-specific characteristics are what determines if changing time is necessary between two recipes. For example: if two ciabatta recipes have the same dough type and baking temperature, no change-over is needed in the forming station or oven. However if one then has a sprinkle decoration pattern while the other does not, that causes change-over time in the sprinkle step. Even though these two ciabatta recipes may be nearly the same otherwise, it is unwise to group them together in a larger group for that reason. As such, we will not group bread types together.

2.5) State of simulation and optimization at Kaak

The engineers and employees at Kaak design and produce their many different machines in-house, and for this they utilize several different mechanic design and machine operating software packages. Also they use software that runs the steel-cutting robots in the machine factory, which we will leave out of consideration. In terms of simulation of on-going processes, a few Siemens software tools are in use such as Simatic-PLC and Mechatronics Concept Designer. These tools are primarily used to simulate the movements of individual machines, and to calibrate the Programmable Logic Controllers (PLC's) that send these machines their commands. For example, these tools are used to direct the movements of the rising chamber, to calibrate the cuts that are made by the cutting robots in the small-bread line, or to direct the movements of conveyor belts. However, these moves are not necessarily optimized, only programmed.

Apart from that, some engineers have dabbled in creating extensive Excel datafiles in which basic calculations are done concerning the total capacity constraints of a single machine. An example of this is a file examining the movements of the scrabbler machine in detail, and another file calculating the proper distance between buns in the cooling spiral. However, as far as the responsible engineers know, no overarching optimization tool exists. In essence, the machines they produce are exceptional at doing their one job, and can function together mostly without problems in a complete machine line if only a single recipe is produced. The machines are designed to run at a certain tact-time, and so if no errors occur, this leads to a stable process.

This all works fine until a customer decides to produce a set of recipes on the same day, which is a common occurrence in several bakeries. Then, as we know, there is no advanced way for Kaak to optimize the process of sequencing jobs. For example, finding a solution to the scheduling of recipes on a single machine, the rising chamber, is a process that's already been on-going for 10 years according to some engineers, and still without definitive answers. Currently all optimization of production schedules is done manually by production managers on-site, using simple heuristics or rules of thumb that may or may not be optimal. Often, they focus on optimizing the throughput of one machine in the line that they see as a bottleneck, such as the rising chamber, without paying attention to the effect this has on the rest of the line, which may lead to sub-optimal results. The phrase *"we've always done it this way"* is likely to be used a lot on these factory floors. A big knowledge-gap in production optimization strategies, such as flow-shop-modelling, is present.

A continuous production line is by no means special in food manufacturing, and using rules-of thumb when only producing a single item is very common. However, a hybrid production line with multiple products, sequence- & machine-dependent change-over times, product-specific production times, no-wait criteria and big continuous batches presents a very complicated problem. It should be no surprise that more expertise is needed.

2.6) Collected data

Through discussions with engineers at Kaak, by poring over footage of machine lines in action, by analyzing ERP data from live factories and by examining existing datasets and factory layouts, a dataset on relevant bread types, machine running times and change-over times has been collected in an Excel file. Due to confidentiality issues, this dataset will only be made available to the examinators of this thesis and the employees at Kaak itself.

Some parts of the data are still subject to debate. For example, the tact time for each recipe is based on an estimation of the maximum possible number of peel boards of buns that can pass through the machines. However, this rate is limited by bottlenecks in the line, that may disappear if the bottleneck machine receives an upgrade or is physically replaced. Since machines are designed to handle a specific production rate, all that may be necessary to raise the production rate of the entire line is to give a specific machine a software update. This type of critical-path analysis may prove to be a worth-while subject of future investigation. It proves to be nigh-impossible to capture in secure data though, so for now we will work with what we have and assume that the different production times are set and reliable per machine. We are still missing reliable data on the pre-rise and packaging steps. We thus exclude these from our scope entirely. The packaging step is very short compared to the cooling/freezing step immediately preceding it, and it does not seem to have significant impacts on the model either way. The pre-rise phase is almost certain to be more impactful than that.

2.7) Conclusion

Kaak's small-bread line is a complex set of machines that work in near-perfect unison when only a single bread type has to be produced. However, when multiple bread types have to be baked on the same day, this exposes a big knowledge-gap in the company. Due to the complexity and variations of the machine line and the large amounts of variability that exists in between recipes, finding good production schedules is a big challenge. This challenge has not been solved consistently within Kaak: often customers are left to solve these scheduling problems themselves, by altering recipes or planning bigger gaps in between batches. Recently some customers repeatedly asked for better scheduling solutions, for example to reduce the average time their ovens are on due to the current high gas prices, and Kaak stands empty-handed.

We call this the Industrial Bakery Scheduling problem. Finding better solutions to these problems could provide instant value to both the engineering team, sales team, and Kaak's customers. We believe the problems faced by Kaak in finding optimal production schedules or machine arrangements can be solved using flow-shop modelling techniques. Specifically, we have to model a way of accurately finding the total makespan of a production plan for different machine arrangements, and then a way of optimizing the sequence of recipes in that production plan. Although some stochastic effects in the machine line are present, the average process length is very reliable. The scheduling problem is too big to solve with a DES model. We opt for a flexible flow-shop model which can give us a ranked list of multiple solutions, which we could later insert in a DES model to vastly reduce the solution space and required calculation time if we so wish. Alternatively we could use this ranked list of possible solutions directly. In the next chapter, we will do a literature review regarding flow-shop modelling and production sequencing.

Chapter 3: Literature review

In this chapter we do a literature review regarding flow-shop modelling and product sequencing to see which techniques we can combine in order to both create an accurate representation of the small-bread line, as well as provide fast solution methods. We will start with a review of literature regarding the model itself, and continue with an overview of construction- and meta- improvement heuristics we will employ. We will also include some notes about the benefits of flexibility in manufacturing.

3.1) Review of flow-shop models

One of the most prominent articles that was written on the topic of product sequencing in production facilities is Graham (1979), whose three-field $\alpha|\beta|\gamma$ problem notation is the standard for operations research to this day. Here α represents the type of machine environment, β represents the job characteristics, and γ represents the optimization criteria. Machine environments can vary from job shops, where various machines are independently present in a factory with products taking whatever sequence they want to be produced, to a flow-shop, where all goods follow the same procedure in the same general sequence of machines.

The case of the small-bread production line is clearly a flow-shop, since all recipes follow the same route along the same machines, even if they don't interact with the machines. Therefor we use the letter *F*, for flow-shop, and *m* for number of machines in our model. *Cmax* stands for the **Max**imum **C**ompletion time, or the time that the last product finishes the last production step. This Cmax value is commonly referred to as makespan, and should be minimized. That will be our optimization criterium. However, describing our model as $Fm|.|Cmax$ is too simplistic.

Reisman et al. (1997) provide a meta-analysis of the field of flow-shop modelling. Flow-shop models have been intensively studied since the 1950's, with at first a focus on constructive heuristics. With the improvement in computers and greater accessibility of computing power, focus has shifted to improvement algorithms and meta-heuristics. Reisman et al. point out that much of the research being done in this sector had little immediate impact on company policy. According to their research, many of the more complicated algorithms that are developed are barely adopted by real manufacturers and are at most influential in academic research. Almost 25 years later, this is not hard to imagine if we look at the situation at Kaak, where specialist knowledge about flow-shop solving methods is severely lacking. A take-away from this is that we should ensure our application is accessible to non-specialists, namely the users at Kaak that will likely end up making decisions based on the outcomes of the model.

According to the framework of Ruiz et al. (2008), our model shares a lot of familiar aspects with the "Hybrid Flexible Flow-shop Problem". This is a flow shop problem where at least one stage has more than one machine (hybrid) and where batches can overtake each other (Flexible). Both factors could apply to our model, although we choose to exclude any possibility of batches overtaking one another: we stick to solving a Hybrid Flow-shop Problem, HFP. Ruiz et al. use a variant of the Newaz-Enscore-Ham heuristic as a prominent construction heuristic in their experiments, where they model the simulation as a Mixed Integer Programme. It should be noted that in the work of AllahVerdi et al. (2006) this terminology is not used in the same way: they seem to use the terms Flexible and Hybrid interchangeably.

Lee and Vairaktarakis (1994) as well as Gupta (1988) prove that HFPs are NP-complete, even in cases where there are only two stages and one of the stages only has one machine. Since our problem has more than 2 stages and at least one stage with two possible machines, it follows that the Bakery HFP

is also NP-complete. As such, complete enumeration to find a single optimal solution is only viable in cases where the number of recipes is low.

A. Allahverdi and Aldowaisan (2001) use a setup-time in their no-wait, 2-stage flow-shop model with sequence dependent-setup time. *"No-wait"* refers to the continuity of jobs for a single product: once the process is started, no waiting time is allowed in between machines. Sequence-dependent setup-times also apply to the small-bread line, since the sequence of recipes has direct influence on the occurrence and duration of change-over times. Interestingly they model the change-over time in their problems as consisting of a setup time, before a job enters a machine, and set-down time afterwards. Our own situation only calls for a setup-time, no set-down time.

Ali Allahverdi et al. (2006) provide a study of 300 different research papers on a wide variety of job-shop and flow-shop scheduling problems, mostly including setup or removal times. Using their notation, which was itself adapted from Graham (1979), our own problem can be described as a hybrid flowshop problem (*HFP)* with multiple stages *(m)* that has anticipatory *(ant)*, sequence dependent Batch setup time *($ST_{sd,b}$)* no waiting in between batches *(noWait)* and with a total makespan-minimization goal (*CMax)*. "Anticipatory sequence-dependent batch setup times" mean that we can start the setup-times before a product enters a production stage, in opposition to for example some assembly jobs where setup-times can only begin once a product has been locked in place in the production area. In the three-field notation our problem can now be described as:

$$HFPm|ant; ST_{sd,b}; noWait|Cmax$$

Some of the papers they cite have a lot in common with our own problem. Specifically the paper by Hall et al. (2003) shares some core characteristics such as the grouping of identical products in big batches to avoid change-over times. However, their solution method concerns "lot streaming": cutting up a big batch of identical products into several smaller lots to make more efficient use of available machine capacity. Crucially, that only applies when machines must process (part of) an entire batch of products at the same time, for example when an oven is filled with buns, closed, and bakes for a set amount of time before being opened. That is not the nature of the Kaak small-bread production line, which in this case allows for a continuous stream of products that are baked independently from one another.

There are examples of continuous flow-shop production systems, for example from the steel industry, such in the research of as Pan (2015) where the subject of cast-scheduling for batches of steel was discussed. However here each individual cast has a processing time per production stage, and the casts are added together in batches called "charges" (functionally: tubs of molten steel to be distributed). No change-over times are discussed, and waiting time is allowed. The biggest size of problems that were discussed concerns a total of 30 casts that were to be scheduled, consisting of different batches. Although the individual casts are functionally independent, it seems the total batch needs to progress to the next machine stage before the previous machine can be declared "free", in each case. That means this problem is not compatible to the Industrial Bakery Scheduling Problem for multiple reasons: the size of the batches, change-over time and waiting requirements.

The Kaak case, in which big batches of different products follow each other on what is in essence a big production belt that runs through different machines with independent change-over times, is a problem that is not commonly described in the literature we have found. As such, we will have to develop a novel calculation model. This could be an interesting addition to the body of existing research in flow-shop modelling. Since it has immediate real-world applications, perhaps it may even have a persistent impact in industry.

3.2) On the efficacy of improvement algorithms

The influential NEH-heuristic by Nawaz et al. (1982) remains a standard in flow-shop construction heuristics. It works by firstly sorting the jobs by individual makespan, and inserting the highest scoring jobs in a production plan in the most favourable position until all jobs are assigned. It is fast, simple to implement, and reliably outperforms other construction heuristics if the number of jobs outnumbers the number of machines. Although in the original paper the heuristic was applied to a simple flow-shop sequencing problem without setup times and only one machine per stage, $Fm|prec|Cmax$, the basic principle can easily be applied to any flow-shop problem. The NEH insertion heuristic performs best out of the techniques Allahverdi & Aldowaisan (2001) used in their set of experiments.

Taillard (1990) compares the NEH and Tabu-search techniques applied to flow-shop modelling questions and finds that Taboo-search outperforms NEH, if it is given enough time to run. Taboo search is a meta-heuristic technique that compiles a list of the solutions it has already checked, and makes them "tabu", meaning they are not checked again. This simple concept helps to avoid the loops and local optima that simple search heuristics often get stuck in, although it is a much more computationally intensive operation that scales exponentially with the number of products that have to re-ordered. Others, such as Ben-Daya and Al-Fawzan (1998) improve upon Taillard's implementation of Tabu search by adding a variable tabu-list and diversification/ intensification schemes and find it out-performs the Simulated Annealing (SA) algorithm in many cases, at the cost of considerable computation time.

The SA algorithm is another improvement meta-heuristic, first proposed as a solution method to the travelling Salesman problem by Kirkpatrick et al. (1983).The method is an analogy to the process of annealing in metallurgy, say, the mix of two molten metals that form stronger connections if the temperature of the melting pot drops slowly. As a heuristic it searches a random solution from the current solution neighborhood, and accepts or denies the solution based on its performance compared to the current solution. If the solution is better it is always accepted, and if the solution is worse it is more likely to be accepted if the "temperature", or acceptance-value of the heuristic is higher. This "temperature" drops along with the number of iterations, excluding more and more bad solutions, hopefully leaving the best ones as an answer to the original problem.

Ruiz and Maroto (2004) perform a study on a variety of constructive, improvement- and meta-heuristics for the permutation flowshop problem. They find NEH to perform the best out of all constructive heuristics, and Tabu-search and Simulated Annealing to out-perform the other meta-heuristics. According to them, other meta-heuristics are too dependent in their performance on the initial solution. We will thus use these solution methods as a basis for solving our Industrial Bakery Scheduling problem.

3.3: Flexibility in manufacturing

Part of this thesis will discuss the benefits of adding more flexible, advanced machine models to the small-bread line. Jordan and Graves (1995) define process flexibility as the ability *"to build different types of products (…) on the same production line".* This clearly applies to the small-bread line, where many different products get produced on the same line. In their research they focus on adding flexibility by allowing more products to be produced on the same line, also called "chaining" or "pairing". In our model, all products can already be produced on the same line, and flexibility is added by installing machines that reduce change-over times in between products. Still, a parallel can be drawn between their research and our own problem: it is often hard to quantify the added benefit of flexibility, as opposed to simply adding production capacity. They conclude that adding

limited flexibility to manufacturing systems can result in more efficient use of available capacity, and reduce the effects of uncertainty.

3.4: Conclusion

In order to solve our Industrial Bakery scheduling problem, we will need to build our own Flow-shop model, which needs to accurately represent the continuous nature of the production line. Due to the novelty of the model itself, we will stick to tried and tested construction and optimalization heuristics: the NEH heuristic for initial construction of solutions, and Tabu-Search & SA for further optimization. We can easily combine these methods as well, by feeding Tabu & SA the outcome of the NEH heuristic as an initial input.

Chapter 4: Developing the goal function

In this chapter we will begin transcribing the complexities of the small-bread line into a deterministic optimization model. We start by discussing our goal function and experimental factors, then the machines that are included in the model, and explain the calculation steps. We build the goal function and discuss the various inputs, and how different machine arrangements can be modelled. In the next chapter we will utilize the goal function to build various construction- and optimization algorithms. In Chapter 6 we will test which one of our optimization methods provides the best solutions.

4.1) Goal function and experimental factors

Our goal is to find a sequence of recipes that results in the smallest possible makespan for any given input. It is unlikely that a production manager would plan more than 12 recipes in the same day. Realistically, the input will be around 5-10 recipes per day, with batch sizes anywhere from 25 to multiple 100's of peel boards. However, if a customer wants to find the optimal sequence of recipes for a multi-day production run, this tool should be able to help them do that. As such we choose not to place an upper limit on the size of the production plans. Apart from the input-production plans, experimental factors also include the different versions and settings for the machines. These different versions and settings will result in different production times, setup times, or machine behaviour. We assume that no matter the settings of the line, any production plan is feasible, but that sub-optimal ones will result in a long makespan.

Let us define our production plan P as follows: a set of any positive number of recipes $r \in R$, coupled with individual batch sizes (in PLBs) $N(r) > 0$, that have to be produced in a single continuous production run. The batches are modelled as single indivisible entities with a beginning and an end. Whilst it is practically unheard of for a production plan to take more than a single work day, we see no reason to put a cap on the maximum duration. The settings of the machines (or factory settings) S are captured in the dataset, of which the main properties will be more carefully explained in Section 4.5.

The idea behind our model is that a central goal function, $F(P, S)$, can be called upon to calculate the makespan of any production plan P in any factory setting S. Depending on the machines in the model, the sequence of recipes and the amount of PLBs being produced of each recipe, it will calculate the total running time of the production plan from first entry into the system to the last exit from the packaging machine. This makespan is the value we try to minimize, with lower values meaning better scores. The algorithm makes these calculations in an additive way, similarly to the model described by A. Allahverdi and Aldowaisan (2001). This means we work from the start to the end, calculating the entry and exit times of recipe 1 on each machine before moving on to recipe 2: the entry/exit times of recipe 2 depend on those of recipe 1, not the other way around.

The result of this Goal function can be fed into several construction or optimization algorithms, which can work until they have found a (locally) optimal solution. We will discuss the construction and optimization strategies in Chapter 5. We build our model in Python 3.8 due to the ease of implementation, visualisation, and low cost of further use by the parent company, although admittedly it may not provide the most optimal running times. A programming language such as C++ may generally be better suited to large combinatorial problems, because it has more efficient data management capabilities, but it would take the author of this thesis much longer to implement any efficient code in it. Regardless, the average problem Kaak faces will likely only require a few simulation runs of perhaps five minutes: this is an acceptable time cost and quick implementation is more important in our case, due to general time constraints.

4.2) Included machines and combined steps

All machines listed in Table 1 and/or Figure 2 will be included in our model, with the exception of the packaging machines. In the case of the packaging machines, we were not able to collect enough data to make a reliable analysis: these machines are often provided by Kaak's competitors and as such data is hard to get. Luckily this step is not commonly experienced to be a bottleneck. If significant data for the pre-rise step is ever found, it can easily be added on to the result of any production plan later, based on the starting time of a batch: if we know the time at which a batch should enter the dough dispenser, we can deduce the earlier time at which it should be mixed.

From our analysis in Chapter 2, it appears there are four major components of the small-bread line that take by far the longest time and thus form the biggest obstacles. These are (1) the rising chamber, (2) the oven and (3 & 4) the cooling & freezing spirals. These are the steps where the buns spend a long time doing little else than being exposed to the conditions in the machines, slowly moving along a conveyor belt or even standing still. The other machines in the small-bread line, by comparison, hardly have an impact in terms of makespan. They continually perform an operation on the dough or bread on a conveyor, whilst passing it on to the next step in the process. This operation has been incorporated into the normal speed at which the product moves along the conveyor belt. Whereas in many flow-shop models the processing time of a step changes if it is skipped, this is not the case for these intermediate steps in the small-bread line.

For example at the cutting station, three peel boards of buns move along the line approximately every 24 seconds. This time has been carefully allocated to ensure that each bun on the peel board receives the cuts within that timeframe. However, with buns that do not receive a cut, the timing is still at three peel boards every 24 seconds: for Kaak it is not worth the effort in engineering to change this behaviour. Since the running time hardly changes, an argument can be made that the cutting station is not an important factor of the optimization procedure, since it has little effect on the production rate. Similarly the sprinkler, dough forming line, scrabbler and fakir hardly change their production time due to changes in recipe. Note that all these steps still require a measure of change-over time between different bread recipes though, and skipping them completely would not serve us well. However they can and will be combined into less complicated blocks for the purposes of the optimization procedure.

The freezing step is interesting, since it uses the exact same system and conveyor belt as the cooling step: the freezer step is not much more than an extension of the cooling step, but colder. When a double cooling line is present, a double freezer line is also present. As such, they can easily be combined into a solid step with different settings depending on the machine line. Thus, the (combined) building blocks of the Goal Function will be as shown in Table 3:

Table 3: Combined Machines in the model

Machines	**Step Name**
DoughDispenser + Bread Former + Panning Point	DoughDis
Rising Chamber (a.k.a. Climate Chambers)	Climas
Turning + Sprinkling + 3-Carrier-Collector + Cutting + Scrabbler	toOven
Oven + Fakir	Oven
Cooling+Freezing	CoolFreeze

Since the objective of our algorithm is to minimize the makespan, combining blocks of machines that have little influence on the makespan will increase the speed of calculations, allowing us to do faster experiments without influencing the quality of solutions.

4.3) Simulation of buns in the goal function

In real life, the bread buns travel along the machines either on PLBs, or in rows after they have been removed from their peel boards. For the purposes of our optimization algorithm though, simulating them as either individual buns or as individual peel boards is not a good idea: each batch can contain hundreds of PLBs, each of which can hold over a hundred buns. Creating an optimization algorithm with the ability to change the placement of 1000+ individual PLBs will needlessly complicate the calculations, since any optimal sequence will aim to reduce change-over times and thus never place the PLBs outside of their natural sequence. Instead, we model each batch as an indivisible entity with a beginning, an end and a tact-time. Thus the sequence of recipes in the production plan will be the sequence in which all their respective PLBs pass through the machine line.

We submit production plans to the goal function in the form of sets of two integers. The first integer represents the number of the recipe, and the second integer represents the number of peel boards that are to be produced of this recipe. As such, a valid production plan would be: [(1, 1)], a single set representing a single peel board of recipe number 1, “Baguette de Tradition”. An equally valid production plan would be [(3,400),(20,60),(19,90),(12,40),(5,240)], with five sets representing 400, 60, 90, 40 and 240 peel boards of recipes “3: Baguette Sensation – Pointed”, “20: Pumpkin Bun”, “19: Panetier-Brown”, “12: Catalan Ciabatta”, and “5: Spelt-Baguette”, respectively. This would also be the sequence in which they enter the simulated production line. To change the sequence of recipes, we could for example exchange the sets (20,60) and (12,40), which would lead to a different sequence in the model, and thus most likely a different makespan. The amount of peel boards in each recipe will be referred to as $N(r)$ from this moment on. It should go without saying that $N(r) > 0$ for any recipe in a production plan.

4.4) Calculation of the goal function

In this section we will define our calculation structure clearly. As mentioned in Section 2.2 we are only interested in the entry & exit times of the first and last peel boards in a batch, as well as the change-over and tact-times. Since the batches are indivisible, all other peel boards in a batch are necessarily based between the first and last one, and move in a predictable manner with at a rate equal to the tact-time of the batch. This causes a set amount of time to pass between the start and end of any batch, based on the tact time and $N(r)$. Based on these values we will formulate a Mixed Integer Programme to clarify the general structure of calculations in Table 4.

Table 4: Mixed Integer Programme

goal	$\min Cmax = \ min\ Lastexit(m_{max}, r_{max})$
	Subject to:
(1)	$FirstEntry(m,r) = 0, \quad r = 1, \quad m = 1$
(2)	$FirstEntry(1,r) = \ FreeMachine(1, r-1, r) + \ PostPoneTime(r), \quad r \in 2, \dots, R$
(3)	$FirstEntry(m,r) \geq FreeMachine(m, r-1, r), \ m \in M, \ r \in R,$ *and not r = m = 1*
(4)	$FirstEntry(m,r) \geq \ FirstExit(m-1, r), \ m \in M, \ r \in R,$ *and not r = m = 1*
(5)	$FirstExit(m,r) = \ FirstEntry(m,r) + \ ProcessingTime(m,r), \ m \in M, r \in R$
(6)	$LastEntry(m,r) = \ FirstEntry(m,r) + \ TactTime(r) * N(r), \quad m \in M, r \in R$
(7)	$LastExit(m,r) = \ FirstExit(m,r) + TactTime(r) * N(r), \quad m \in M, r \in R$
(8)	$FreeMachine(m, r, r+1) =$ $\quad LastEntry(m,r)$ if no change-over $\quad LastExit(m,r) + \ ChangeOver(m,r,r+1)$ if change-over.
(9)	$PostPoneTime(r) \geq 0 \quad r \in R$

Goal: Minimization of total makespan. Equal to the last exit of the last peel board from the last machine.

1. First batch starts at time 0. Note that this does *not* include pre-rise times.
2. Ensures that each subsequent batch will start production once the previous batch has been cleared, and when the planned postponement-time has passed
3. Ensures that each batch can only enter a second machine after this machine is ready for the next batch
4. No waiting times allowed in between machines, batches keep moving continuously from start to finish.
5. The first exit out of any machine occurs one processing time after the first entry.
6. The last peel board entry into a machine occurs after the entire batch entered this step
7. The last exit out of any machine occurs one processing time after the last entry
8. The time a machine becomes free after processing a batch
9. All postponement times must be positive

The postponement time in constraint (2) is the exact amount of time between the finalization of change-over time on the dough-dispenser and the entry of the next batch of bread onto the line. This time is required to avoid waiting times on the rest of the line, which cannot stand still as per constraint (4). However to get accurate, minimal postponement times, first we have to figure out what waiting times would occur on the line without them. We will further discuss this in Section 4.6. Note that this calculation structure results in a single outcome per production plan, per list of machine settings. As such, this is a deterministic formula that can be calculated quickly, but which has a large amount of possible inputs and an equal number of possible outputs. Whilst the number of machines in the model is known, the number of recipes can be any number $R > 0$.

Let us discuss an extreme example, to see how this calculation structure performs in practice. In both Figure 4 and Figure 5 a production plan with a single recipe is produced. The production plans are: [(1,10)] and [(1,1000)]. Recipe 1, “Baguette de Tradition”, is the same but in Figure 4 the batch size is only 10 peel boards. In Figure 5, the batch size is 1000 peel boards. In both cases the makespan starts at T = 0, and ends after the last peel board rolls off the last production belt. The bars represent the total time a batch is processed per relevant machine group: the first processing time starting immediately after first entry, and the last processing time starting directly after last entry. The brightly colored parts of the bar in the top-left and bottom-right corners of each bar represent the timespan in which PLBs of this batch enter and exit this machine respectively, whilst the dark parts represent the processing time of the first and last PLB. This means that for Figure 4 the first of ten peel boards is finished at time 184.1 after the start of production, whilst the last is finished a little more than a minute later at 185.6 minutes. For Figure 5 we can see that the first of 1000 peel boards is also finished at 184.1 minutes, whilst the last rolls off the production line over two full hours later at 334.1 minutes.

As such, it is clear that the size of a batch has no influence over the time it takes the first peel board in a batch to complete the production run, whilst it has immense impact on the timing of the last peel board. However, as should be clear from Figure 4, small batches can still take up space in a machine for a long time regardless of their length, because they can have considerable processing time and may not be combined with a succeeding batch. It is also clear from this example that, because the tact-time and batch-size are constant throughout the calculations of each recipe, the entry-time and exit time are of the same length, and match up with the previous step: as soon as the first peel board exits one machine, it will start entering the next machine. This overlap is very clear to see in Figure 5.

From these graphs it should also be clear why we chose to combine multiple intermediate machines into a single “DoughDis” and “toOven” group: even combined, the steps have little impact on total makespan. If we did not apply this combination, Figure 4 and Figure 5 would have been roughly 3x as long whilst conveying the exact same information.

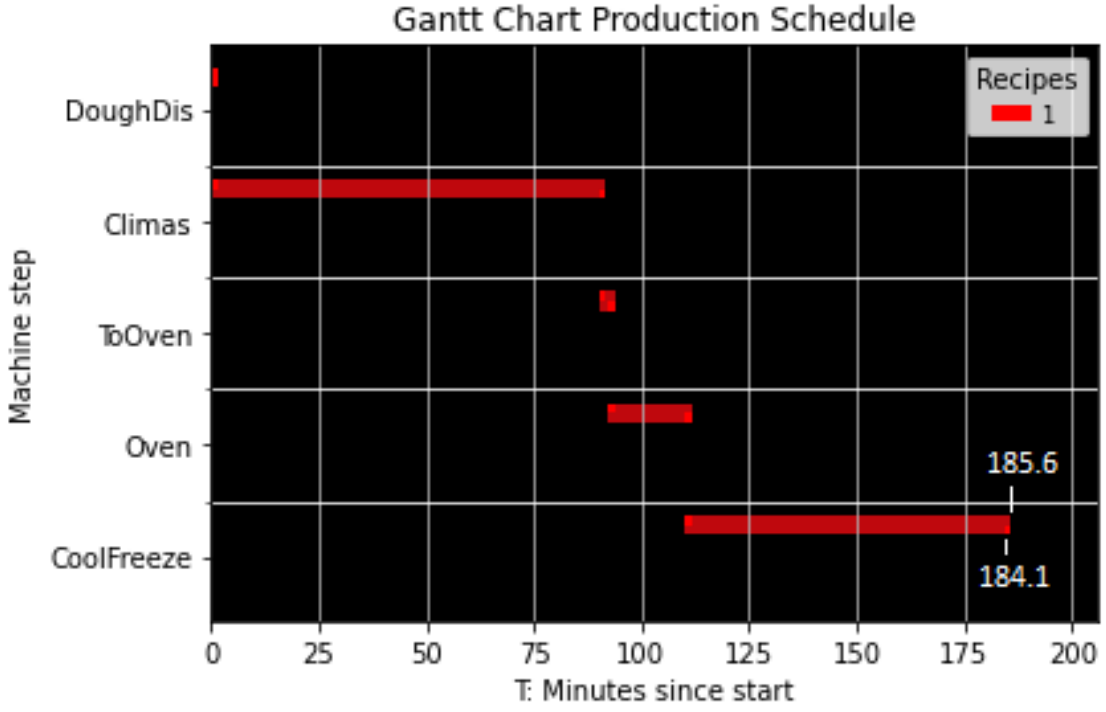

Figure 4: Gantt Chart of 10 peel boards, combined graph of transitions & processing time

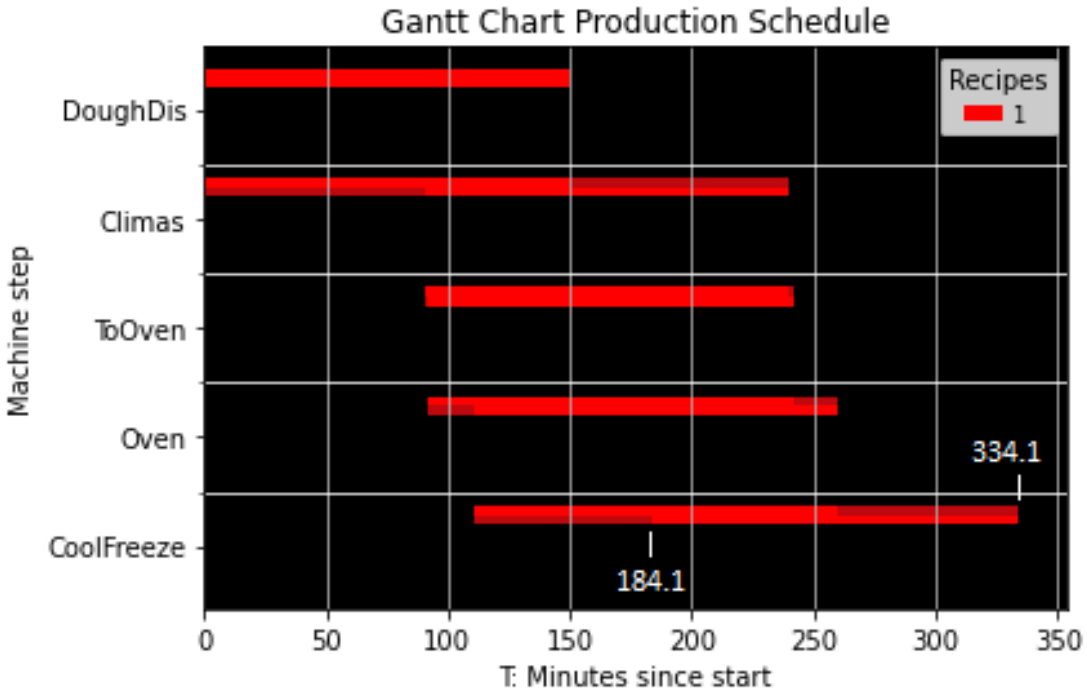

Figure 5: Gantt Chart of 1000 peel boards, combined graph of transitions & processing time

4.5) Change-over times

The machines in the model can either be free, processing a batch, or changing over between batches. Two or more batches can be processed at the same time, if no change-over time is present between them: for example, two batches of bread that are to be baked at the same temperature for the same amount of time can follow each other "head to tail" on the oven conveyor belt. Since we know the tact time $TT(r)$, the Processing time $P(r,m)$ per machine and the number of PLBs per batch $N(r)$, all that remains is the Change-Over time $CO(m,r,r+1)$. This change-over time is dependent on the machine in question and the combination of recipes. For example in the Kaak case, if a brown bread recipe is processed before a white bread recipe in the dough dispenser, the machine needs to be cleaned afterwards to ensure no contamination takes place, which increases change-over time. When the white bun comes before the brown bun, no extra cleaning is needed. Therefore, $CO(m,r,r+1) \neq CO(m,r+1,r)$. This sequence-dependent setup time will result in a matrix of $R * R$ values per machine group in the line, which is saved in the dataset.

Let's assume a dough has finished the pre-rise step, and is the first to enter the system. The first PLBs of a batch will then enter at time $T = 0 = FirstEntry(m,r)$, and exit at time $FirstExit(m,r) = FirstEntry(m,r) + P(m,r)$, the processing time of a peel board. The last peel board of this batch will enter the machine at $LastEntry(m,r) = FirstEntry(m,r) + TT(r) * N(r)$, as $N(r)$ PLBs need to be produced before this last one can. This last peel board will exit the machine one processing time later, $LastExit(m,r) = LastEntry(m,r) + P(m,r)$, at which point the machine enters the change-over phase of length $CO(m,r,r+1)$ after which the machine becomes free, $FR(m,r)$.

Note that if $CO(m,r,r+1) = 0$, the next batch can immediately start entering a machine after $LastEntry(m,r)$, directly following the previous batch. So then $FR(m,r) = LastEntry(m,r)$. This is essentially what happens whenever two peel boards of the same batch follow each other through the machines: the model is very efficient in dealing with large identical batches, because it condenses all intermediate products into a quick and simple calculation. A visual representation of the calculation structure without postponement times is shown in Figure 6. We will deal with the implementation of postponement times in Section 4.6.

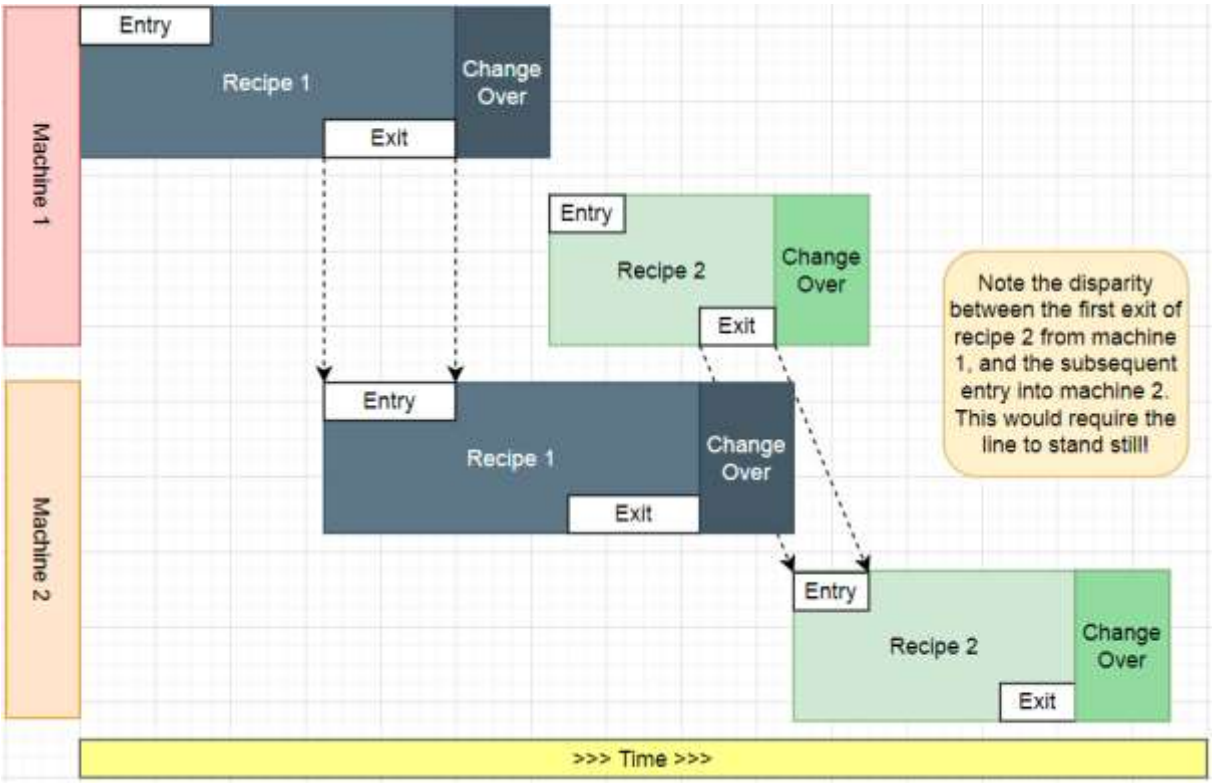

Figure 6: Gantt Chart of two recipes on two machines, with (forbidden) waiting times. Each "recipe" bar represents the time that any of the peel boards that make up this recipe are present on this machine.

This same general structure of calculations is repeated for every step in the production process, but some exceptions exist. The running time of the Goal function depends on the number of recipes and the number of included machines. Because there are 5 values that have to be calculated for every recipe for every machine, the calculation time of the Goal function increases linearly along with the increase in recipes. We denote this running time of the Goal function as roughly: $T(v) = \sim m * r * 5$. Roughly, because depending on the settings of the model some steps can take slightly longer to calculate than others.

4.6) Waiting on the line and postponement times

Any time a machine is done with a production step and has passed the change-over time, it is opened up for the next batch of products. In working this way, we push a new batch of bread onto the line as soon as the first machine, the dough dispenser, is free. This can incur waiting times for recipes on the processes in between the first and last machine, if those have longer waiting times than the dough dispenser. Dough that cannot move through the line continuously would be negatively impacted in terms of rising/cooling times. This is the function of the $noWait$ clause of our optimization problem: we cannot allow our bread to wait on the line. An efficient solution to this would be to calculate an extra waiting time for all batches before they start their production process, which can easily be factored into the starting time of a batch. This way, we essentially shift the entire batch forward in time to eliminate waiting times during the production process. We call this preventative waiting time the postponement time. Of course, we want this postponement time to be minimal, but in order to find out what the minimal allowable postponement time is, we will first need to calculate the waiting times that would otherwise occur on the line.

To solve this problem, our goal function goes through two phases per recipe when calculating the makespan of a production plan. Firstly, it calculates *only* the first-entry and first-exit times per stage. From these, the required waiting time for the next machine is deducted. We add this time to

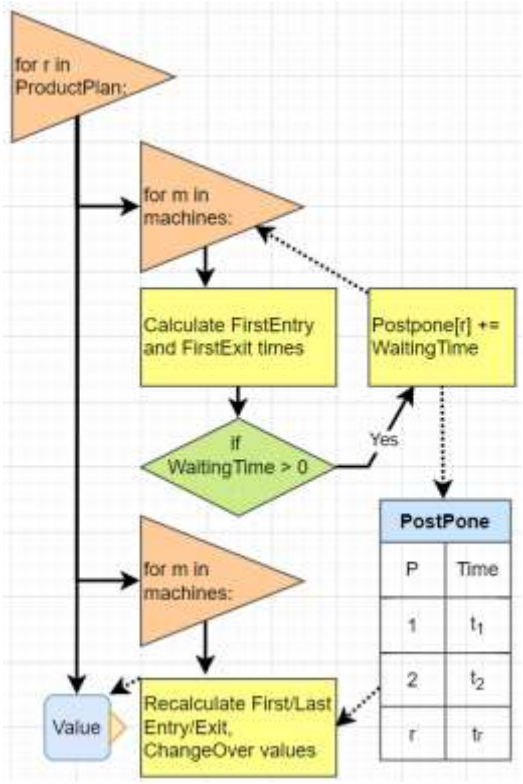

Figure 7: Postponement time calculation

$PostPoneTime(r)$, incorporate the waiting time into the starting time for the next stage, and repeat this for all stages. This results in a single postponement time for the recipe. That concludes phase one. In phase two, the true timings are calculated for this recipe. We incorporate the postponement time into the starting time at the dough dispenser and set accurate entry and exit times for the recipe in each stage. This ensures no internal waiting times for this batch are present. We then move on to the next batch, and start again in stage 1. We can now make use of the accurate timings of the first recipe to make our first-entry and first-exit calculations for the second.

The result is a makespan and a set of postponement-times, to be added to the waiting time at the start of production of each batch. This function takes roughly 1.15 times as long as calculating the Goal function normally without incorporating the *noWait* clause. This small delay is deemed acceptable, and we will continue using the function in the rest of this thesis. A simplified visual representation of this process is shown in Figure 7, accompanied by a Gantt chart of a simple two-stage problem in Figure 8. When compared to Figure 6, this second Gantt Chart clearly shows the impact that extra postponement time can have on the elimination of waiting times from the model.

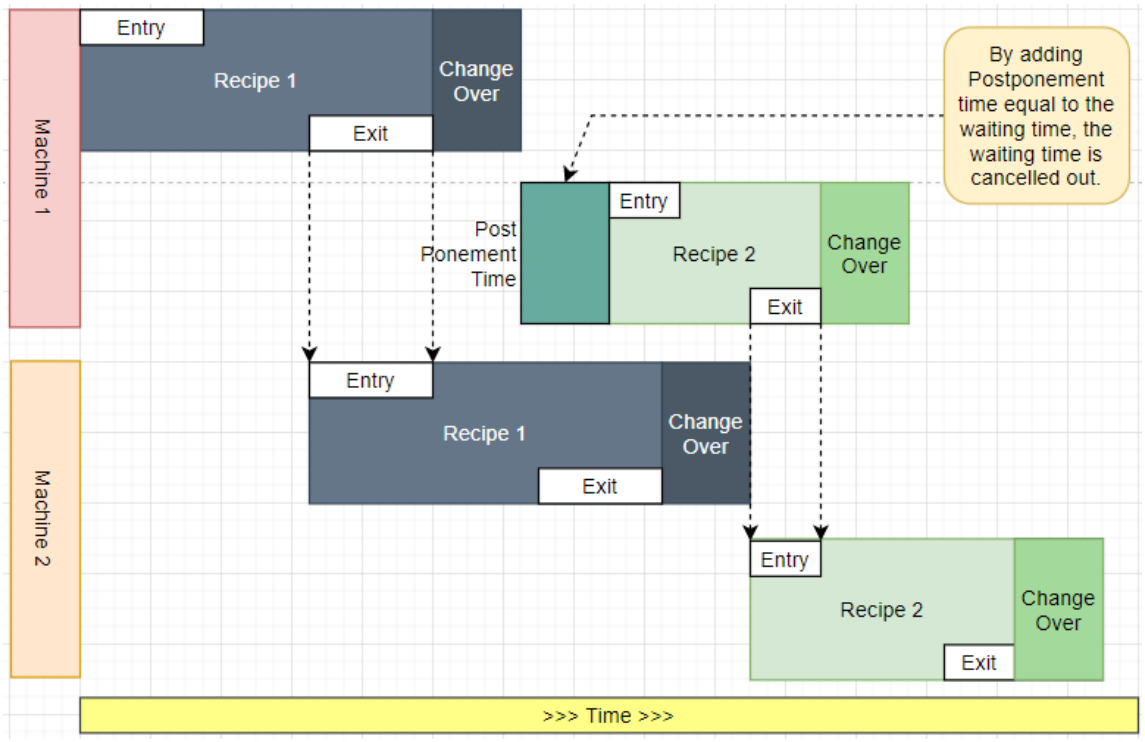

Figure 8: Gantt Chart of two recipes on two machines including postponement time

4.7) Settings and machine versions

Settings for the multitude of machines and recipes can be entered and saved in an Excel data-file which can be easily accessed and changed by Kaak. These include, for example, the heating and cooling potential of the oven, or the cleaning time required for the dough dispenser. To save processing time during the optimalization run, the effect of the interaction effects between different recipes will be pre-calculated. If a recipe ever needs to be added, the list can simply be extended. Also if the temperature at which a bun is baked is changed, or if the heating potential of the oven is increased in the model settings, this will instantly affect the change-over time for the oven step. The goal here is to ensure that the tool remains usable after the thesis is finished, whilst also saving time calculating these otherwise unchanging values.

As an example, say we have two recipes that follow each other into the oven. For recipe A, the required baking temperature is 180°C, whilst recipe B calls for 210°C. If the heating and cooling potential of the oven is 5°C and 2°C per minute respectively, then the time required to change the temperature would be 6 minutes if B followed A, or 15 minutes if A followed B. This is a simple linear example but if Kaak wished to go more in depth with complex warming formulas, we could easily change the formula in the dataset. The same goes for any of the other change-over values.

Whilst the general calculation structure, as outlined in the goal function, holds true in the basic version of the machine line, some machines have variants that cause exceptions. These machines are the rising chamber, oven, and cooling & freezer spirals. The basic variant of the machines is most commonly found in the small-bread lines, but in the interest of justifying investment into the more advanced variants, the performance impact of non-standard machines is very important to both Kaak and its customers. As such, we integrate them as optional into the goal function. They can be altered in the settings in the excel program, or as part of a run of experiments. Essentially, we can turn these settings ON or OFF, with a simple True or False statement.

Overall, the advanced machine models provide the same level of continuous production, but help save time by eliminating or alleviating waiting time caused by change-overs or occupied machines. They add flexibility to the model, which should correspond to a better use of available production capacity in accordance with Jordan and Graves (1995).

Rising Chamber

The rising chamber has a variant where instead of one big chamber, multiple smaller chambers stand side by side. That allows for some more flexibility in the usage of this machine. Two consecutive recipes with different rising times could never follow on the regular variant, since the regular machine can only run at one speed throughout the machine and thus the rising time of at least one of the recipes could not be achieved. Therefore in the regular variant, a recipe needs to exit the rising chamber completely before the speed of the tower can be changed. The variant with multiple towers allows a recipe to bypass a number of towers to cut down on a portion of the total rising time, without a change in speed. This saves especially much time if the difference between resting times is small: in a case where the first recipe has a resting time of 2 hours, and another 1.5 hours, this would cause two hours of waiting time in a normal rising chamber, which would translate to a maximum of 2 hours extra postponement time. In a resting chamber with four towers only half an hour of waiting occurs, after which the second recipe could skip the first tower and fall in line directly behind the first recipe: no change in speed is necessary in the other 3 towers to make the total processing time 1.5 hours. In the advanced version, the formula used in calculations of the rising chamber changes to:

$$FreeMachine(m,r,r+1) = LastEntry(m,r) + Max(ProcessingTime(m,r) - ProcessingTime(m,r+1),0), \qquad m = Climas$$

In Figure 9 and Figure 10 the effect is displayed graphically, although not to scale, with a regular rising chamber and a rising chamber with multiple towers. Note that more than two towers are possible, and that the towers do not necessarily have the same size: the most common variant of this more advanced machine is a variant where one tower runs at 30 minutes, whilst the next runs at 60 minutes. This allows much more flexibility than a single tower that runs for 90 minutes. Functionally this does not impact the formula compared to a model with, for example, three towers of 30 minutes, and as such we can safely speak of two versions for the rising chamber.

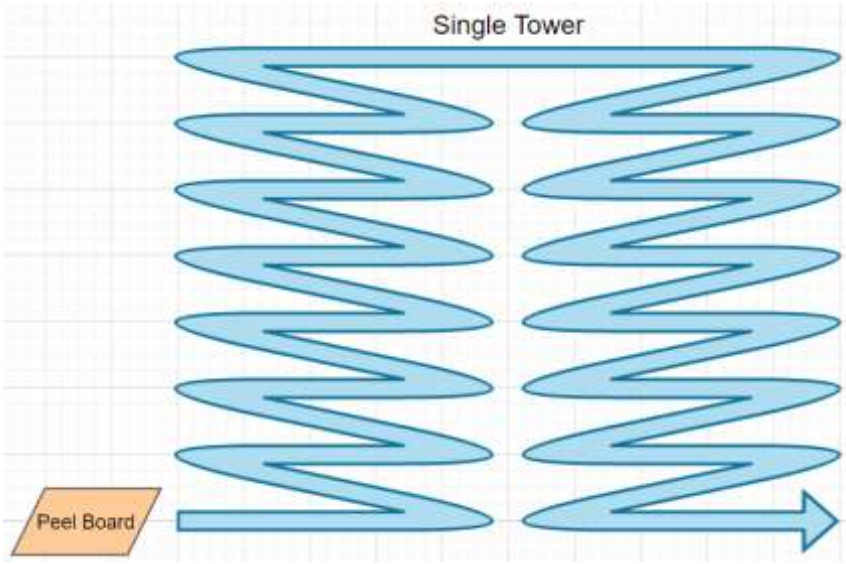

Figure 9: Single Rising Chamber, low flexibility

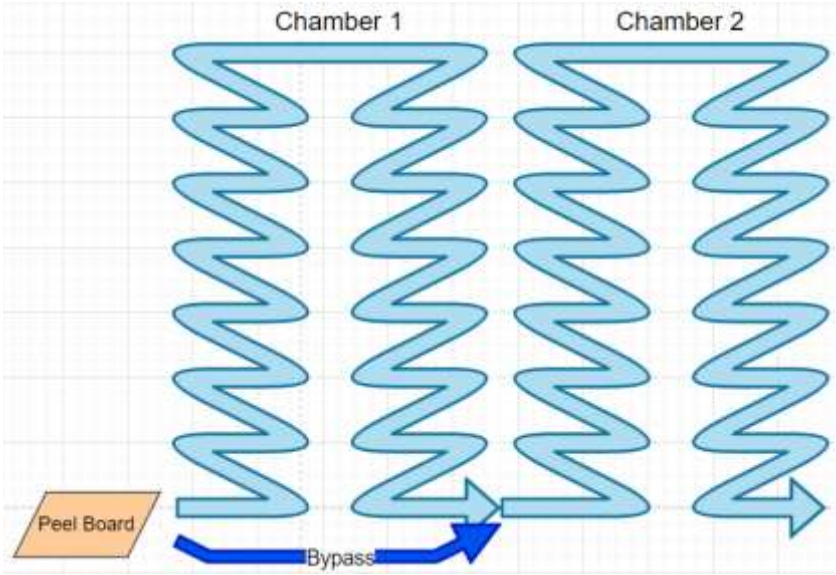

Figure 10: Double Rising chamber, with bypass for recipes with low rising time. High flexibility.

Oven Line

The oven line is essentially a long, heat-proof conveyor belt that passes through a straight oven. Change-over times consist of two parts: the time needed to cool down or warm up the oven in between two recipes with different baking temperatures, and the speed of the belt. Two recipes with the same baking time and temperature can follow each other directly. Two recipes with different

baking times need more space, since one batch needs to exit the oven completely before the speed of the oven-belt can be changed. This change is nearly instantaneous. In case of a temperature difference, all bread needs to finish baking and exit the oven before the temperature can be changed. This temperature change is not instantaneous. The oven-variant with separate heating zones, also called Oven Partition, will allow the zones of the oven to start changing temperature as soon as the first batch has left that zone, meaning the change-over process can be started whilst bread is still in the oven.

This is a major time saver in some cases: it can mean 75% of the baking time is subtracted from the change-over time if there are four oven-zones. However, oven partition has no impact on change over caused by bread types with different baking times, since it does nothing to affect the speed of the belt. Oven partition is not a standard part of the ovens that Kaak offers as of yet, but Kaak is very interested in testing the efficacy of such modules and as such we will add it to the model. A version with four different oven zones is seen as a likely candidate by Kaak engineers, so that is the example we will pick for our model. Theoretically, the oven could be divide into hundreds of segments. See Figure 11 for a comparison of the normal oven and an oven divided in 4 heating zones.

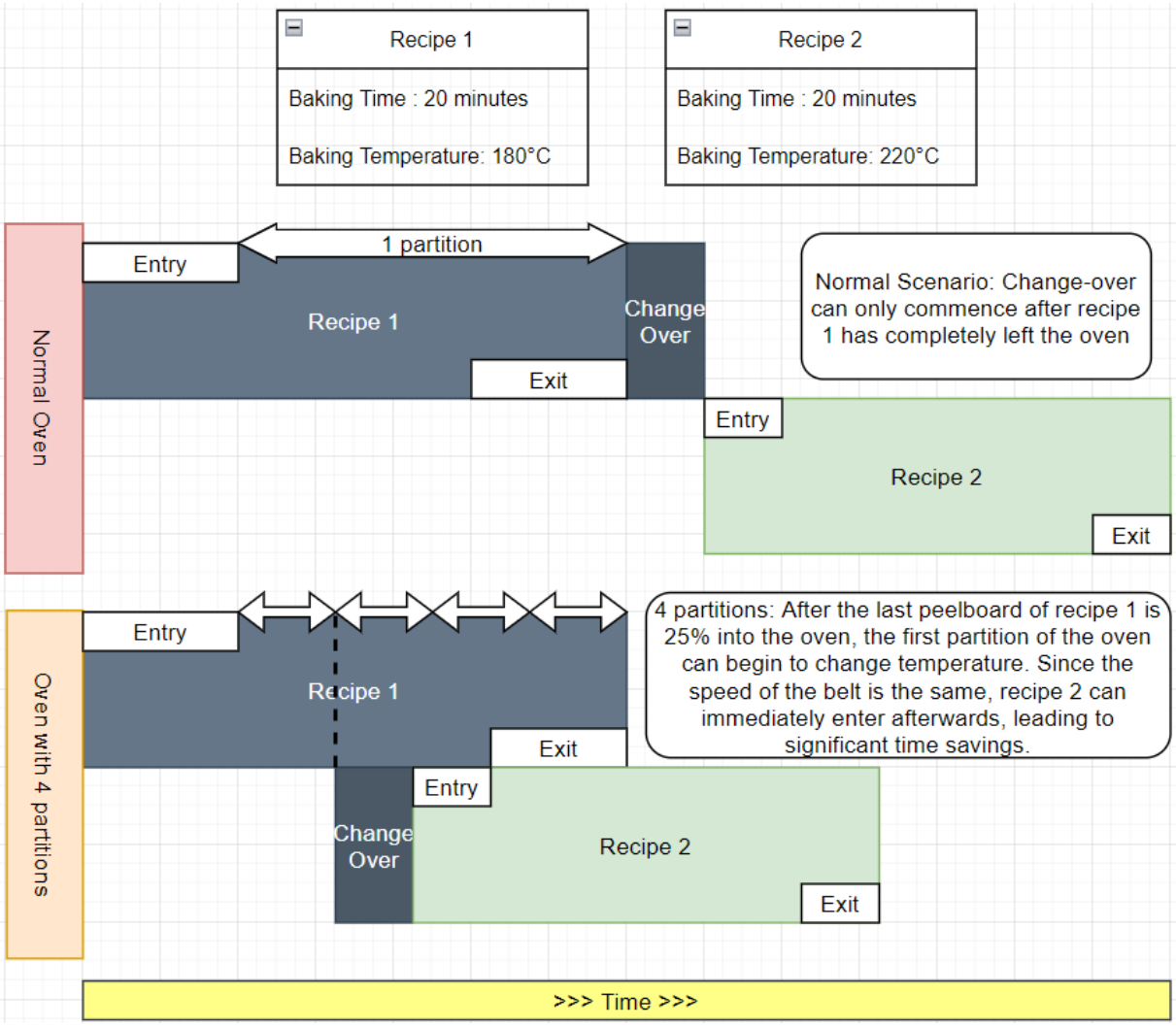

Figure 11: Gantt Charts of two recipes in a normal oven, and an oven divided in 4 partitions

In the model, the formula changes as follows:

$$FreeMachine(m,r,r+1) = LastEntry(m,r) + \frac{P(m,r)}{4} + ChangeOver(m,r,r+1) \quad \text{if } P(m,r) = P(m,r+1) \text{ for } m = oven$$

$$= Max(LastEntry(m,r) + \frac{P(m,r)}{4} + ChangeOver(m,r,r+1), LastExit(m,r)) \quad \text{if } P(m,r) \,!= P(m,r+1) \text{ for } m = oven$$

Cooling Spiral

The cooling spiral has two different versions: essentially, a version with one and two independent belts that can move at different speeds. The speed of the belt is dependent on the length of the cooling belt, and the required cooling time per recipe. As such it is always known in advance. The cooling process can take an hour or more, and two recipes with different cooling times cannot follow each other head-to-tail on a single belt. This is because if the speed of the belt changes, then so does the area of the belt that becomes available per peel board of buns. The buns must lay in a pre-determined pattern and this pattern cannot be achieved if less space is available. Therefore it is often the case that a batch must wait for the entire previous batch to leave the cooling belt before being allowed to enter, even though the belt is empty at the front. This causes huge waiting times and delays, which translates to an increase in postponement times at the start of a batch.

The double-belted version of the cooling line solves this problem by sending the second batch of bread onto the different belt, which has an independent processing speed. We model this by allowing two recipes with the same cooling time to follow each other head to tail, whilst sending other recipes to the second belt. If a third batch with (again) a different cooling time follows whilst both belts are occupied, it then has to wait until either of the belts free up. Note that this double cooling spiral is the feature that makes our flow shop model "Hybrid", as described in Section 3.1. The rest of the model consists only of single machines. We can set a certain tolerance for acceptable cooling time in the dataset, since often in real-life cases bakers will ignore or otherwise alter their production guidelines to allow non-ideal batches to follow closely on the cooling belt. Note that with the simple rules we set in the usage of this double cooling belt, the outcome of the model is still deterministic. Although total makespan is influenced by the choice that is made, these rules will lead to minimal makespan per sequence in the plan.

Freezing Spiral

Finally the freezing module is also closely linked to the cooling spiral. It is essentially the same machine but ramping downwards, even sharing the same belt. If the freezer is enabled, we model this by doubling the required cooling time for any recipe. When a double cooler is installed, a double freezer is also present and it does not add any further complications to the model.

As such there are four machines which cause considerable variation in the duration of the makespan, all four with two variants. The rising chamber, with either one or multiple towers, the oven with or without partition (we will assume partition into four equal parts), the cooling belt either with or without a twin, and a freezer line which either is or is not present. These four machines with two options each result in a total of $2^4 = 16$ possible versions of this machine line, easily enabled or disabled by changing the settings in the Excel file. Luckily this set of machine variants is small enough that we can still perform a complete test on all of them within a reasonable time frame.

4.8: Conclusion

In this chapter we created a novel calculation method for the continuous multi-stage hybrid flowshop problem with anticipatory, sequence dependent batch setup time, no waiting in between batches and with a total makespan-minimization goal. Our calculation method immediately assigns minimal postponement-times per batch, to ensure no waiting times will occur on the line. It is specifically modelled to correspond with the Industrial Bakery Scheduling problem that Kaak faces, although it can easily be modified to correspond to other continuous flow-shop systems. We now have a functional way of calculating the makespan value of any production plan on any of 16 machine arrangements without waiting times in between steps. The model generally calculates the makespan value of any set of recipes at an average of 0.000333 seconds per recipe, or 3000 recipes per second. So, if a production plan of 10 recipes would need to be optimized, our Goal function could give the valuation of roughly 300 variants of this production plan per second. With this, we can now start to solve the scheduling problem.

Chapter 5: Optimization of the Goal Function

Now that we have a functional way of calculating the makespan of any input of production plan and machine settings, we can begin the process of optimizing these inputs. In this chapter we will discuss our implementation of the Nawaz-Enscore-Ham heuristic and two well-known meta-heuristics: Tabu-Search and Simulated Annealing. These techniques have been tried and tested in numerous studies relating to flow-shop modelling, as seen in Section 3.2, and we feel they should be a good fit for our problems as well.

5.1) NEH Heuristic

The Nawaz-Enscore-Ham (NEH) heuristic is a constructive heuristic for Flow-Shop models, as seen in Chapter 3. Our model is a little more complicated than the model described by Nawaz et al. (1982), since they do not take change-over times into account, accept waiting times, and consider each job to be a single entity (they have no problem with hundreds of peel boards closely following each other). Luckily, implementing a variant of this cheapest-insertion heuristic is quite simple when using the Goal function. We create three empty lists: two of which will contain a production plan and an associated makespan value, and one of which holds the final production plan. First, we enter each individual recipe and associated batch size in the production plan into the Goal function separately, and save the output together with the input in one of the lists. Essentially, this is the value of this single recipe as an individual production plan $Fm|n = 1$. This single-recipe production plan does not have any waiting- or postponement times, and so it is the minimal amount of time that any recipe will take to be baked. We sort this list from high to low makespans, leading to a sorted list of the value of the individual batches. See Figure 12 for a visual representation of the value calculation of these singular recipes. Note that this will result in the same sorted list of values for any initial production plan containing the same recipes and associated batch sizes.

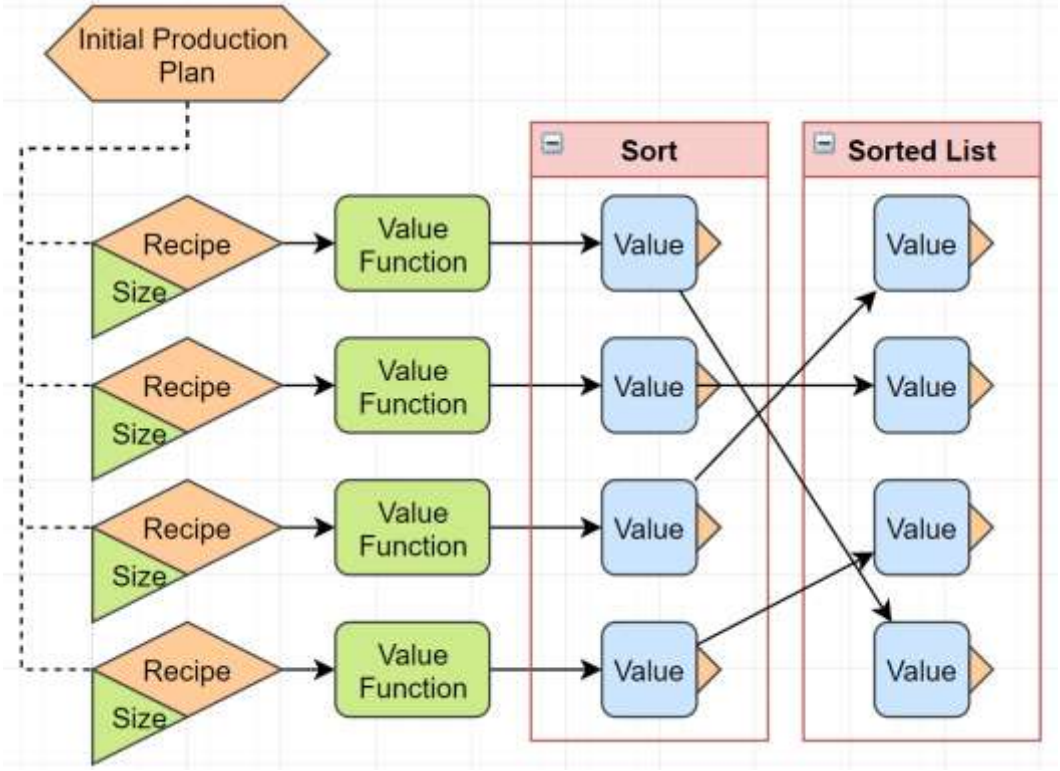

Figure 12: NEH Heuristic, first stage - single recipe makespan value calculation, and sorting

We place the first batch of this list, the individual batch with the highest makespan, in the final production plan. Now, we take the second recipe on the sorted list, and temporarily place it in all possible locations in the final list, checking the value after each placement. For the second recipe, this leads to two production plans that are saved to the second sorted list, as it can be placed either before or after the first recipe. We pick the production plan with the lowest timespan, empty the second sorted list, and repeat this procedure for the rest of the recipes in the first sorted list. So recipe three is placed at three different places in the final production plan, the best is picked, etc. Finally, this leads to a full production plan in which all recipes have been placed. See Figure 13 for a visual representation of this second part of the heuristic. This simple construction method significantly improves the value of a random initial solution and is very likely to find the optimal production plan when the number of recipes is small. It finishes in $R + R * (R + 1)/2$ moves, often within one second for problems with 20 recipes.

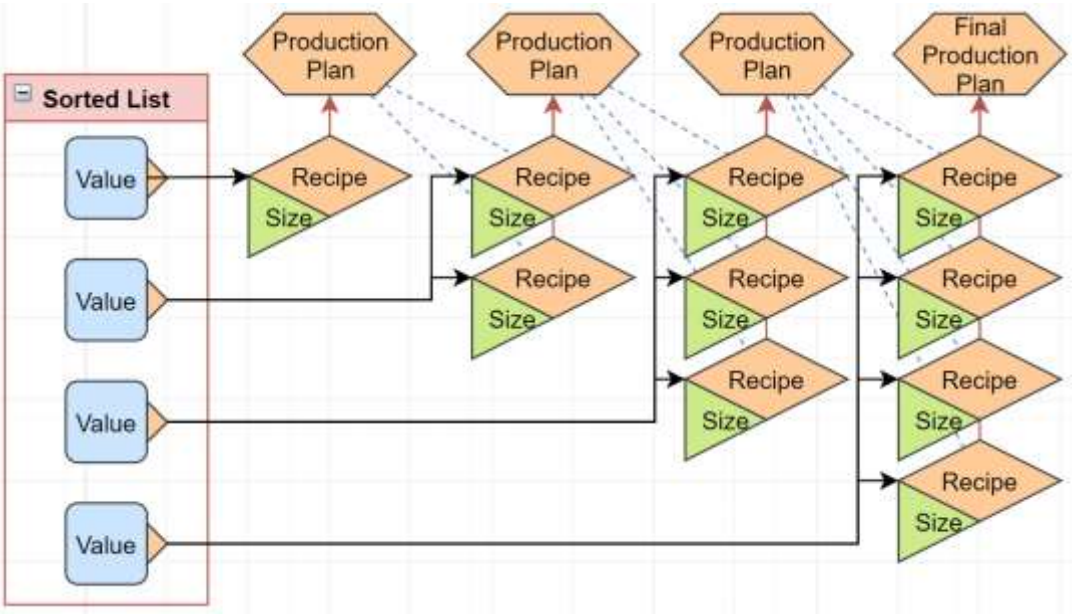

Figure 13: NEH Heuristic, second stage – production plan construction

5.2) Tabu Search

Tabu-search, as explained in Section 3.2, is an improvement meta-heuristic that is commonly used to solve combinatorial problems, such as the Flow-Shop optimization problem. For our Tabu function, we experimented with the use of two separate local-search heuristics: a delete-insert heuristic, and a 2-swap heuristic. Both local search functions create a list of at maximum $R * (R - 1)/2$ solutions, excluding the solutions that are already present on the tabu-list. The tabu-function then picks the best solution from this list that is not on the tabu-list. If it is better than a known solution, it is inserted in the list of known best solutions. That list of best solutions is then trimmed if it is too long. Either way, the chosen solution is placed on the Tabu-list, which also gets trimmed if it becomes longer than the allowed tabu-list size.

We use a dynamic tabu-list size in this algorithm: if a new better solution has been found, the tabu-list decreases in size by 1, to a minimum of 5. On the other hand, if no better solution gets found, the tabu-list increases by 1 to a maximum of $(2.5 * R)$. The function terminates after a set number of iterations, or after it encounters a solution that is already on the list of best solutions combined with the same length of tabu-list. That is a clear sign that the function has started looping around the same set of solutions, which would make it pointless to continue. In some small-scale experiments the delete-insert search outperforms the 2-swap search by a significant margin, of around 3% of the

makespan value in a series of 20 experiments, where in all but one cases tabu found a better solution with the delete-insert heuristic than the 2-swap heuristic. This aligns with the findings in the research of Taillard (1990) and other papers. Therefor we conduct the rest of our experiments with the delete-insert option for Tabu-search.

5.3) Simulated Annealing

Simulated Annealing, as was briefly introduced in Section 3.3, is an improvement meta-heuristic that is often used for a wide variety of optimization problems. It is originally based on the cooling mechanics of annealing metals. Our version works with both a 2-swap and delete-insert local search function, which show similar results. We also added a useful 2-swap local search function with the ability to lock recipes in place, which could prove useful if a manager needs to finish a recipe first or last and still needs to find an optimal sequence. This "lock" function essentially takes one or more recipes out of the pool of recipes that are to be swapped, and places them back in position afterwards.

Our algorithm follows the common stratagem for SA, with a starting temperature and Markov-chain length of certain size, and an alpha-value $[0 < \alpha < 1]$ which slowly decreases the temperature. In order to prevent confusion inside Kaak, a company that famously works with many different oven models operating at numerous heat levels, we will refer to this "temperature" as "Acceptance Factor" (AF).

In Simulated Annealing an initial production plan is used as input of the local search function, which randomly selects a new solution in the neighbourhood of the input. The value of this production plan is then checked in the goal function. If the value is better (lower) than the current solution, it is accepted as a new production plan outright. If the value is better than that of any known solution so far, it is inserted in the list of best solutions. That list is then trimmed if necessary. So far it works just like a random exploitative function.

If the value of the newly found production plan (NV) is not better than the old value (OV), we check the difference in value, divided by the current AF. This value is used as the exponential of *e*, always resulting in a value between 0 and 1, and the resulting number is then checked against a random number uniformly drawn between [0 , 1]. If the value of the calculated number exceeds the random draw, we accept the new solution and make it our new starting point. This results in a higher average acceptance rate of bad solutions if the AF is high, and if the difference between the two values is small.

$$Accept\ if: rand(0,1) < e^{\frac{OV-NV}{AF}}$$

An iteration consists of a set number of solution draws, leading either to acceptance or rejection. After a number of draws dictated by the Markov-Chain, the acceptance factor is then altered with:

$$AF = \alpha * AF$$

This then leads to a higher amount of exploration early in the algorithm, and higher amount of exploitation later in the algorithm. The algorithm ends if the AF drops to a pre-set level, if it runs out of iterations, or if it has added a new solution to the list of best solutions in a number of iterations. We start the algorithm with an AF of 25, a Markov-chain value of 50, and an alpha value of $\alpha = 0.985$. The maximum number of iterations is 250, the number of iterations between an improved solution is set at 2x the number of recipes. The minimum pre-set level of the AF is set at 0.04. Note that with the current alpha value, the algorithm never reaches this AF: at maximum it will drop to $25 * 0.985\text{^}250 = 0.0446$. This would still be a relevant stopping criterium if the Alpha value is

changed though. This seems to provide a good balance between running time and exploration of the solution space.

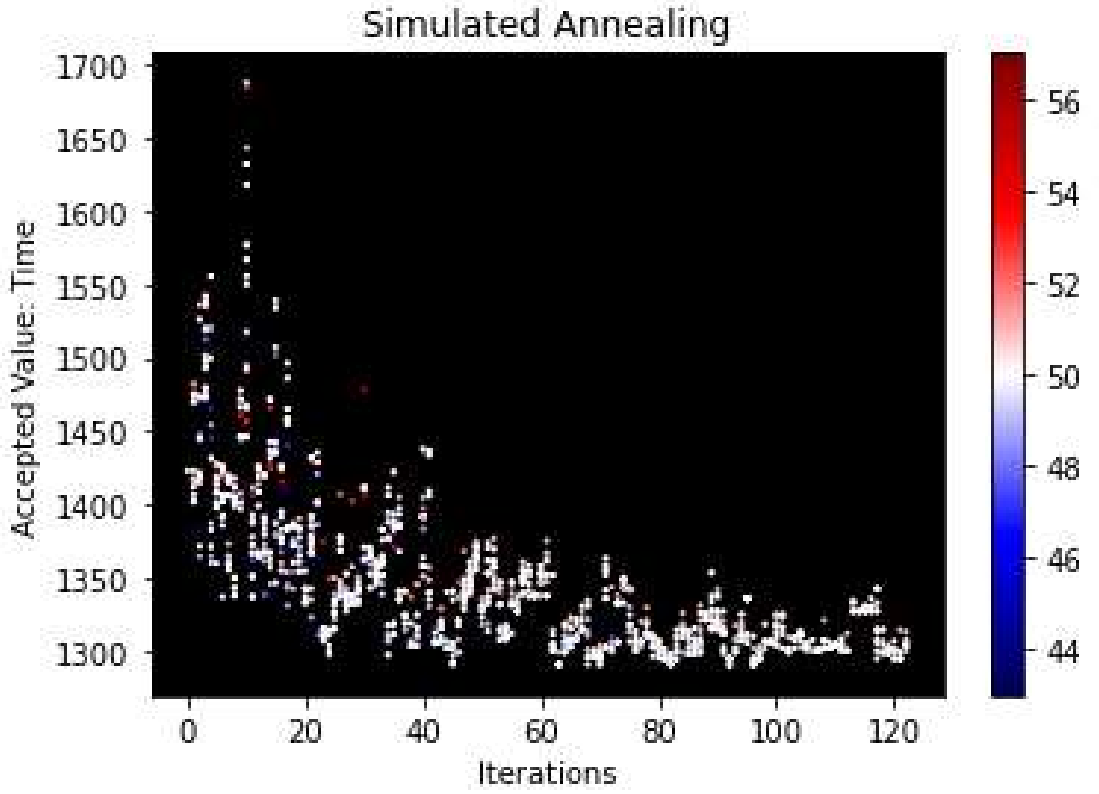

Figure 14: Scatter plot of solutions explored by Simulated Annealing algorithm in experiment with 15 recipes

In Figure 14, a clear difference can be seen in the amount of exploration vs. exploitation at the start and end of the iterations. Note that each individual iteration consists of a set of draws, which is why the dots form vertical lines in the plot. Red dots appear when a newly accepted makespan was higher than the previous, blue appear when the new makespan is lower than the previous. White answers are more neutral. In this experiment, the algorithm concluded way before the maximum number of 250 iterations was reached. This is because the algorithm failed to add a better solution to the solution list for 30 iterations, which is two times the number of recipes included in this test.

5.4) Conclusion

In this chapter we developed three distinct optimization methods for our value function. One for initial construction of a valid solution, and two for further optimization. We can now begin with experiments to validate the model and test the methods.

Chapter 6: Validation and Experimentation

In this chapter we will firstly validate the model we created by solving a representative customer support case, to see if the model is a good representation of a real-world system and if it gives satisfactory answers to the questions customers are likely to ask. Secondly, we will run a series of randomized experiments with varying amounts of recipes in the production plans, to test the performance of our different solution methods. This will allow us to optimize the tool for further use, by utilizing the best solution methods for certain types of problems and eliminating unnecessary trials. Finally, we will run a second set of experiments with production plans of the same amount of recipes, but with different batch sizes. This last set of experiments will allow us to see the effects that varying machine models have on average makespan, both individually and in combination with other machines.

6.1) Model Validation

An example case of a set of production plans and corresponding customer questions were shared with us by a Kaak employee, which we translated and transcribed below. The production plan concerns a weekly production target, which is to be averaged out over the different days. It is reasonable to assume that this bread factory will take an average of one day per week off to do proper maintenance and cleaning of machines, and as such our daily production target can be safely regarded as the weekly production target divided by six. The case revolves around a factory with a standard dough dispenser, an advanced rising chamber, a normal oven, a single cooling tower and no freezer. We can model this by enabling the advanced rising chamber in our standard model. The questions regard a change of production plans for the weeks around Easter, when extra types of bread are in demand. The normal plan consisted of five recipes, but around Easter two extra recipes are added to the plan. First, let us look at the (translated) production plan, as seen in Table 5. Here the two extra recipes, specifically for Easter, are added in row 6 and 7.

Table 5: Recipes and quantities for example case, 1-5 for normal plan, 6-7 for Easter.

Recipe nr.	Recipe Name	Amount	Amount/plb	plb/week	plb/day
1.	Baguette tradition	100,000	20	5000	834
2.	Demi Baguette	150,000	44	3410	569
3.	Ciabatta 90g	125,000	88	1421	237
4.	Ciabatta 300g	70,000	27	2593	433
5.	Pumpkin Seed Bun	50,000	63	794	133
6.	Easter Bread	25,000	11	2273	379
7.	Brioche Bun	45,000	40	1125	188

The production times for these recipes are as follows:

Table 6: Production times for recipes in example case

	Line capacity [plb/hour]	Turner	Rising Time [min]	Baking Time [min]	Baking Temp [°C]	Cooling Time [min]
1.	400	No	90	18	240	70
2.	400	No	90	14	240	70
3.	400	Yes	90	12	220	70
4.	400	No	90	15	220	70
5.	400	Yes	90	13	230	70
6.	400	No	60	30	220	70
7.	400	No	60	12	235	70

The customer had the following questions regarding the initial and secondary situation respectively:

Initial situation:

- What is the optimal sequence for this production plan?
- What would the total makespan for this sequence be?
- Which machines have the biggest influence on this plan?
- What is the ratio between productive and unproductive time?

Secondary situation:

- What would be the best place to add the two recipes?
- If we want to compensate the extra time in the second scenario, how many demi-baguettes should we cut from the plan? (Note that the choice for demi-baguettes appears to be random, but this is representative of real-life questions)
- What is the ratio between productive and unproductive time? How does this compare to scenario 1?
- If we want to finish the Easter Bread (Paasbrood) first, how much time would this cost compared to the optimal sequence? Which other viable sequence would push it forward as much as possible?

Additionally, we will investigate which, if any, machine upgrades we should recommend in the second case.

Initial Situation

For the primary situation, the best solution was found by both the tabu-search and simulated annealing methods; the five recipes should be placed in sequence (1, 2, 5, 3, 4), with a makespan value of 562.15 minutes. Figure 15 displays the Gantt-Chart of this production plan.

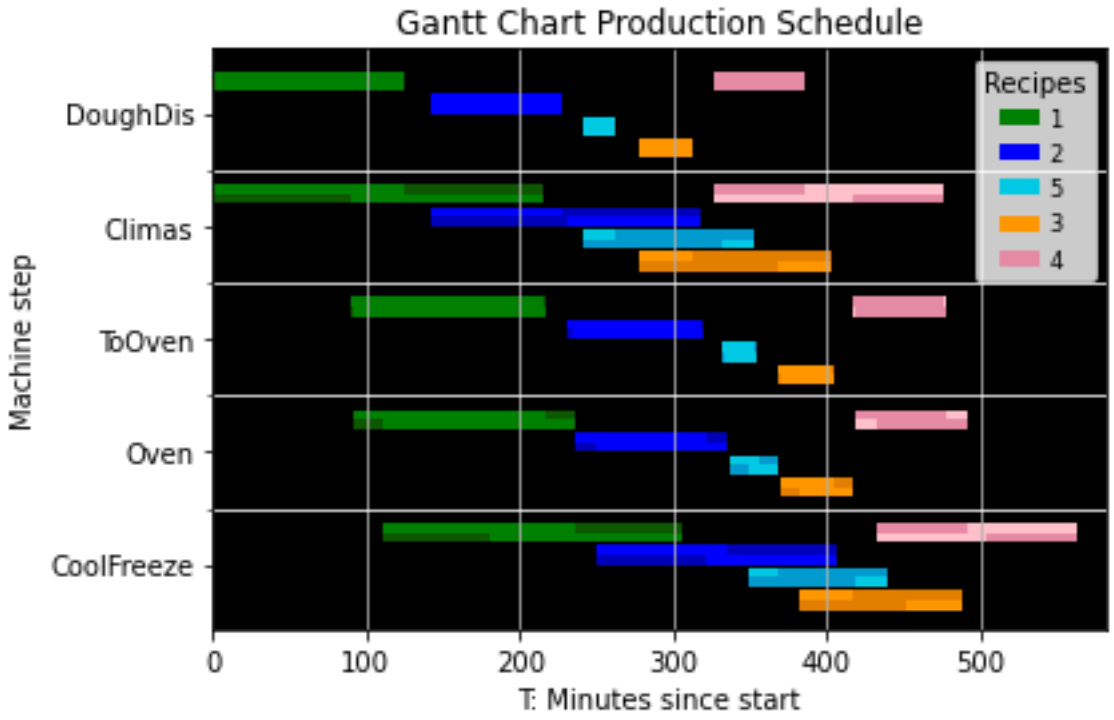

Figure 15: Gantt Chart of optimal initial sequence, combined chart of processing time and entry/exit times

The ratio of productive and unproductive time has to be properly defined: if we define “productive time” as the time where any of the machines in the line are actively processing products, then in this production plan there would be 100% productive time, since there is no overlap of unproductive times across all machines. However in this production system that would always be the case, since each recipe is always active in at least one machine due to the continuous nature of the system. As such, this is a bad definition.

If we measure productivity per machine, this will give us more accurate data on the time a machine spends actively processing buns. However “activity” itself is not a good indicator of productivity either: if two batches of 1 peel board with different cooling times follow each other in the cooling tower, the change-over time may be negligible after batch 1 has left the belt, and the machine may be processing a single peel board 100% of the time, but this is clearly not an efficient use of available machine capacity. We therefor decide to measure productive time by comparing the active time in the dough dispensing step to the total time the dough dispenser is used. Any delays in this step point to inefficiencies further down the line that have to be solved with extra postponement times. The dough dispensing step has low processing times, and small batches cannot keep it occupied for long stretches of time: by measuring the performance of this step, we will actually measure the rate at which new peel boards enter the system. Combining this metric with the makespan of a production plan will be a much better indicator of overall planning efficiency.

In the first experiment with five recipes, the total time the dough dispenser was active was 325.74 minutes, and the last bun exited this step 385.15 minutes after the start of production. This leads to a productive time of 84.6% in the dough dispenser. This means that 15.4% of the activity around the dough dispenser was used for either change-over operations or postponement time.

The machine with the biggest impact on this production plan is the oven. In the dataset that was provided, the rising and cooling time are equal for the first five recipes. The baking time and temperature were different for each recipe. This can be clearly seen in Figure 15 where the climas and cool-freeze steps show significant overlap between subsequent recipes, while the oven has no overlap, and minor change-over times.

Secondary Situation

We added the two easter recipes to the production plan. The best solution was found by the SA algorithm with a makespan value of 687.85 minutes, which is 125.7 minutes longer than the previous production plan. The best sequence would be (7,6,1,2,5,3,4), adding recipes 7 and 6 to the start of the plan, although this solution is tied with several others. The ratio of productive and unproductive time is 80.4% in the dough dispenser. If we would want to cut a number of demi-baguettes to compensate for the extra two hours of production time, we would need to cut them all from the plan to reach a minimum makespan value of 587.5 minutes. This means that cutting the demi-baguettes alone is not enough to compensate the extra production time of the easter recipes.

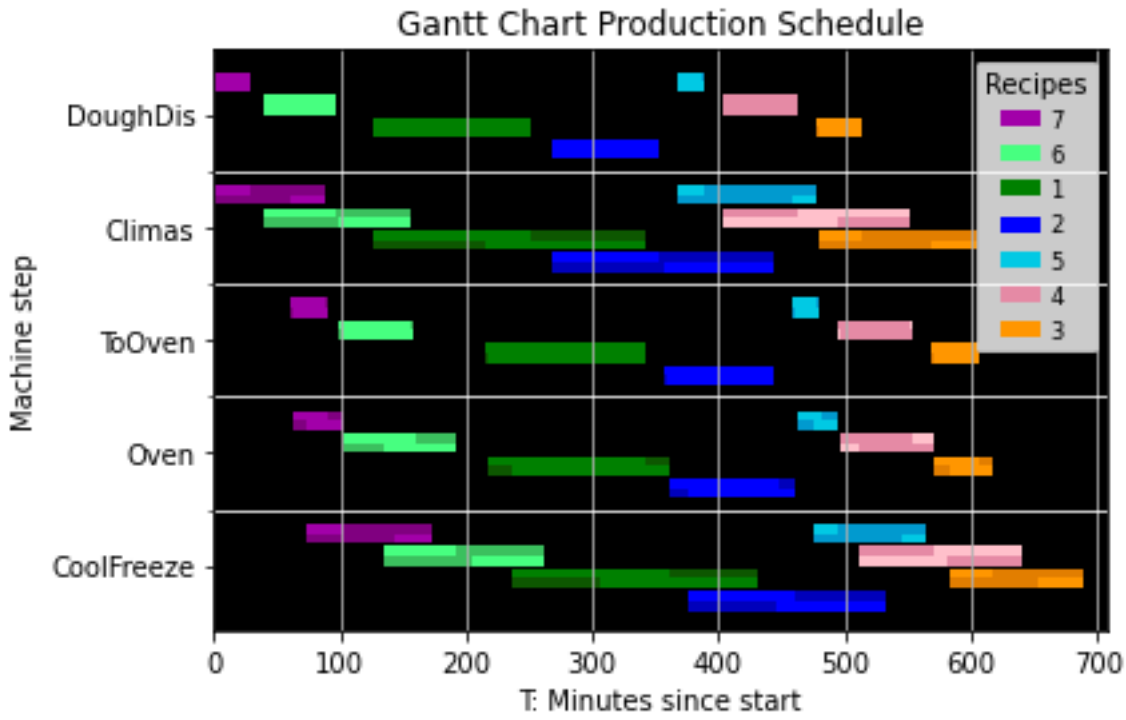

Figure 16: Gantt Chart of optimal secondary sequence, combined chart of processing time and entry/exit times

To answer the question about putting recipe 6, “Easter Bread”, first, we make use of the 2-swap lock function for Simulated Annealing. We place recipe 6 first in the production plan manually, lock it in place, and let the algorithm find alternative solutions. It found three equal solution candidates, with product sequences (6,1,2,5,3,4,7), (6,1,2,5,4,3,7) and (6,2,1,5,4,3,7) respectively. The makespan for these three solutions is 696.7 minutes, which is 9.85 minutes longer than the real optimum in which the Easter bread is placed second. Interestingly, recipe 7 gets shifted to the end of the sequence, whereas it would have been first in the optimal sequence. This is likely due to its short rising time, which would have been difficult to fit in between other batches.

Finally, seeing as the main obstacle to achieving a lower makespan appears to be the oven module, we will test out the oven partition variant with four separate heating zones. In this setup the new best found sequence is (6,1,2,4,3,5,7) or (6,2,1,4,3,5,7), both with a makespan of 678.7 minutes. This is a difference of only 9.15 minutes with the original optimum, although it does place recipe 6 first in the queue as the customer requested. It is interesting that by upgrading a machine, the most efficient sequence of recipes changes to another: this underlines why it was hard for Kaak to predict the effect that different machine models had on their production plans. Even if a customer was previously working with an optimal production plan, a change in machines can lead to a different sequence being optimal, meaning that the true benefit of upgrading the machine was unlikely to be discovered. A final Gantt chart of one of these sequences is shown in Figure 17. Note the overlap in the production time in the oven for recipes 3, 5 and 7, which have the same baking time but different baking temperatures (see Table 6): this is the change enabled by oven partition. Oven partition has a positive effect on this production plan, but perhaps not enough to warrant an upgrade of the entire oven module. We will leave that to the judgment of the customer.

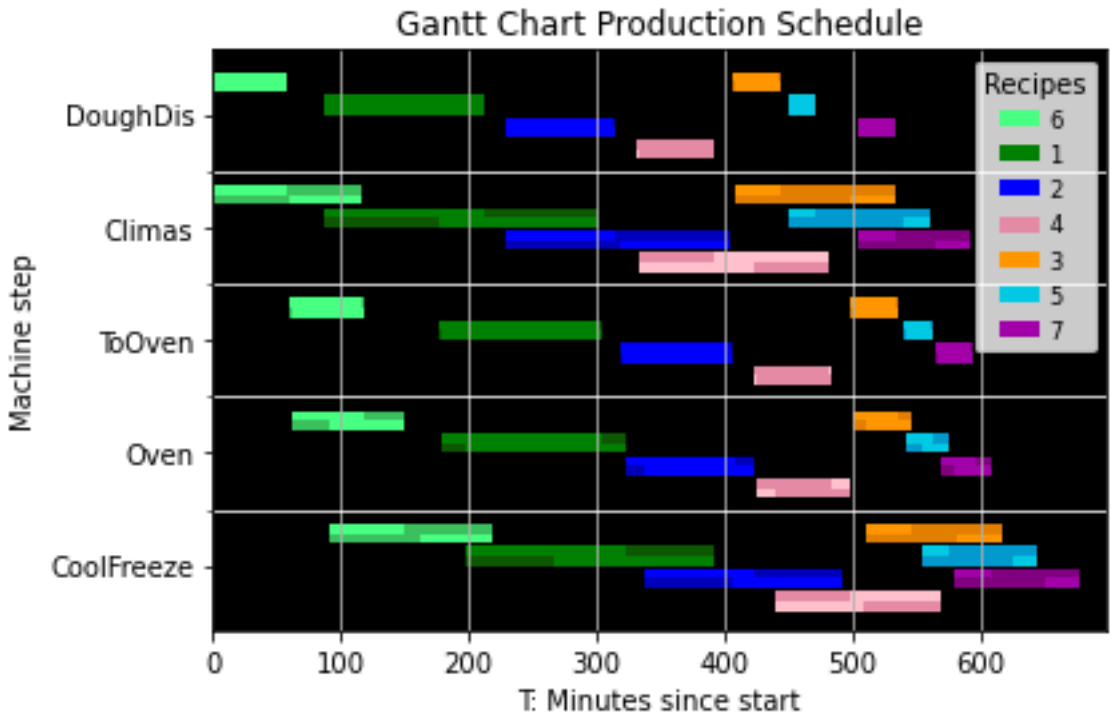

Figure 17: Gantt Chart of optimal secondary sequence with oven partition, combined chart of processing time and entry/exit times

We believe this small test accurately portrays the ease of use of the tool, and the assurance it will give a user of the accuracy of the provided solutions. After the dataset was updated, the running time of the simulations was only slightly over 5 seconds. This type of flow-shop model, let alone the accompanying optimization procedures, were non-existent within Kaak before. As such it represents a significant improvement over the rules-of-thumb that would normally be used to answer such customer questions.

6.2) Randomized Experiments

The randomized experiments will take the following form, loosely inspired on the set of experiments by Ruiz and Maroto (2004). We create a list of randomized production plans, and test each plan on all 16 different versions of the machine line, for all different solution methods. The best answers will be gathered, as well as the time taken to reach each solution. The output of the following solution methods will be gathered:

- Primary Input
- NEH
- Tabu Search
- Simulated Annealing
- NEH + Tabu Search
- NEH + Simulated Annealing

The primary input is the outcome of the input production plan, without alterations to the initial random sequence. This will set a clear benchmark for the other experiment factors. With the NEH, Tabu and SA solutions, we take the initial input and feed that into the aforementioned algorithms. The NEH + Tabu/SA solutions are solutions where we take the initial production plan and construct a better solution with the NEH heuristic, before feeding that into the aforementioned meta-heuristics.

We will test these solution methods on a set of production plans that will be randomly generated, within the boundaries set in Table 7. We insert smaller average batch sizes for production plans with many recipes, because production managers will try to fill up a working day, and no more than a working day, if they can. A production plan with many recipes would therefor generally only have small batch sizes, and vice versa. This is an attempt to ensure the production plans come as close as possible to realistic scenarios. Again, note that the initial sequence of recipes in these samples is random. That means that statistically, at least $1/6^{th}$ of the recipes in the set of production plans with three recipes are already optimal.

Table 7: Experiment 1, setup and repetitions

#of recipes in Production Plan	#of Peel boards per Recipe	#Repetitions
3	[300 - 600]	100
7	[150 - 450]	80
10	[120 - 300]	50
15	[90 - 240]	25
20	[60-180]	10
Total Samples:		265

This list of production plans will be unaltered, but be applied onto the different machine lines. Thus, we will perform 265*16 = 4,240 individual experiments in the first experiment run. From these experiments we will save the following data:

For each machine-setup, for each production plan:

- For each solution method, the best solution value
- For each solution method, running time of the algorithm

This will then allow us to determine the best-performing solution method per amount of recipes. We define the best solution as the solution with the lowest makespan. In cases where two methods find the same best solution, the one with the fastest time gets precedence.

In addition to this first experiment with different amounts of recipes and batch sizes, we also run a second experiment for production plans that all have 10 recipes, but big differences in batch size. This will allow us to clearly see the effectiveness of the different machine models when dealing with different batch sizes. In this second experiment run we will use the optimization methods that have proven most efficient according to the first experiment. See Table 8 for an overview of the second set of experiments. As with experiment 1, we will save the best answers per solution method.

Table 8: Experiment 2, setup and repetitions

#of recipes in Production Plan	#of Peel boards per Recipe	#Repetitions	Name
10	[60 - 180]	50	10S
10	[120 - 300]	50	10M
10	[300 - 600]	50	10L

6.3) Performance of solution methods

After the first run of experiments we can take a look at some interesting performance data. First, let us focus on the effectiveness of the different solution methods, before turning to machine performance.

A clear division exists in the solution method that comes up with the best solution depending on the amount of recipes. (see Table 9 for an overview). In production plans with three recipes, only six possible solutions exist. One or more of those six can be optimal, and since the initial solution is randomized we see almost 1/5th of the initial solutions are already optimal. Excluding those cases, the most prevalent solution method was the NEH heuristic, followed closely by the tabu methods. Tabu will always find the fastest solution in these small solution spaces if NEH fails to do so, and is much faster than SA in these cases because it excludes previously visited solutions. After a maximum of six iterations, tabu search is done, whilst simulated annealing processes these solutions for multiple iterations before realizing it has already checked them multiple times.

Table 9: Experimental Results – Amount of times the best (or best & fastest) answer was provided by this method

#recipes	Init	NEH	Tabu	SA	NEH+Tabu	NEH+SA	Total
3	308	1106	103	0	83	0	1600
7	0	487	177	173	275	168	1280
10	0	162	101	169	182	186	800
15	0	36	64	58	100	62	320
20	0	7	38	23	60	32	160
Total	**308**	**1,798**	**483**	**423**	**700**	**448**	**4160**

If we extend the production plans to 7 or more recipes, the balance begins to shift. Whilst it still comes up with the best solution in 488/1,280 times, the NEH heuristic gives way to a relatively even mix of Tabu, SA, NEH+Tabu and NEH+SA solutions. A notable outlier is the NEH+Tabu heuristic, which seems to excel past the performance of normal Tabu-solutions in every experiment.

It is important to also see the average time spent per solution in this context. See Table 10 for an overview of the average calculation cost per result. Tabu search in particular seems to spend a lot of time calculating when more recipes come into play. This is logical: our Tabu-search function creates a list of all neighbors and picks the best one, and also keeps a tabu-list that is linearly influenced by the amount of recipes in the production plan. In other words, its calculation costs increase exponentially as more recipes need to be placed in a correct sequence. Simulated annealing on the other hand takes far less long in the final experiments, because it has a set number of iterations, and each iteration is no more computationally intensive than it would be in small solution sets: it always picks a single random neighbor instead of initializing the entire neighborhood. We could fiddle around with the settings of the simulated annealing algorithm to make it run a little longer, and hopefully find the best answer more often.

Table 10: Experimental results: Average running time of this method per number of recipes in seconds, rounded to 3 decimals

#Time	Init	NEH	Tabu	SA	NEH+Tabu	NEH+SA	Total
3	0.001	0.005	0.014	0.306	0.014	0.365	0.705
7	0.002	0.041	1.738	2.012	3.197	2.062	9.054
10	0.003	0.111	11.062	5.088	13.057	4.714	34.035
15	0.004	0.364	60.896	14.493	63.996	12.071	151.824
20	0.006	0.845	184.310	34.840	181.993	23.452	425.446
Total	**0.016**	**1.365**	**258.020**	**56.739**	**262.258**	**42.665**	**621.064**

Is the calculation cost of the Tabu function for big batches excessive? If calculating a single solution of 20 recipes costs 0,006 seconds, calculating for an average of 180 seconds would mean that roughly 30,000 solutions were checked within that time. For a solution size of (20!), that is by no means excessive or unexpected. If we assume that for each iteration, n*(n-1)/2 checks of the goal function are performed in order to pick the best, that comes down to roughly 158 iterations. Whilst testing a great set of experiments like in this thesis may take days, an experiment time of several minutes should not pose a problem for Kaak if they ever use this program to solve a singular customer question. Especially since cases with 20 recipes are not likely to show up often.

To really see the difference in performance of the respective function, it may be more prudent to look at the degree of separation between the best-found solution and others. If one method found the best makespan, and another method found a slightly higher makespan in only 10% of the time, that might be of influence on the method we end up adopting for big simulation runs, like experiment 2. In Table 11 this value is displayed as an average of the results from all solutions that were found. For example, that means that on average, the NEH+SA method found an answer that was only 0.5% higher than the actual best makespan in all experiments with 10 recipes. This includes the experiments where it actually found the best makespan, or where it was tied with others.

Table 11: Average deviation from best-found answer

#Recipes	**Init**	**NEH**	**Tabu**	**SA**	**NEH+Tabu**	**NEH+SA**
3	6.3%	0.4%	0.0%	0.0%	0.0%	0.0%
7	21.8%	2.6%	1.5%	0.3%	0.7%	0.1%
10	31.4%	4.3%	2.5%	0.9%	1.4%	0.5%
15	39.7%	5.1%	3.1%	2.0%	1.4%	0.9%
20	47.8%	5.3%	4.0%	2.6%	2.3%	1.6%

In fact if we look at this metric, the NEH+SA method has a more reliable average performance than all the other options we studied, whilst taking the least time out of all the meta-heuristics. For this reason we use SA and NEH+SA as the solution method for our second experiment run. However for solving normal customer questions, like in 6.1, we see no reason to exclude the Tabu-search heuristics from our tool box, since in those cases it will only add minutes to total solution time, and may come up with better results. Since every set of experiments was performed on the same grouping of production plans, the results are not independent. Whilst there is often a difference between the optimal sequence because of the changing machine line, production times and often change-over times do stay the same between different experiments.

6.4) Machine arrangements and interaction effects

We performed 16 runs of experiments with the different machine setups. With this set of experiments we perform a simple ANOVA test concerning the overall impact that different machine arrangements have on the average makespan, as well as the possible interaction effects between machines. First we will address the average outcomes per number of recipes, then the effects of the individual machines, then the interaction effects. We will also show some of the same data collected from the second experiment run, for context.

Firstly, let us look at Table 12 where the results of experiment 1 are displayed. We took the average of the best solutions that were found for each individual production run, regardless of the solution method. Again, note that each machine arrangement was tested on the same set of production plans and so these results should be representative of the performance of the entire line. How to read this

table: Run 1 was the run with no advanced variant for the climate closet, no oven partition, no double cooling spiral, and no freezer. The average result of the production runs with three batches on this line was a makespan of 481.93 minutes. In run 2, we added a freezer to the line, which lengthens the cooling step and in this case resulted in an average optimal makespan of 641.24 minutes for the same set of production plans with three recipes. The average makespan of all production plans with three recipes on al 16 line varieties is 503.09 minutes.

Table 12: Average makespan in minutes of experiment according to number of recipes and machine variants, experiment 1

Run	Climas	Oven_P	D_Cooler	Freezer	3	7	10	15	20
1	FALSE	FALSE	FALSE	FALSE	481.93	674.81	731.66	876.35	916.22
2	FALSE	FALSE	FALSE	TRUE	641.24	871.81	907.98	1029.04	1058.02
3	FALSE	FALSE	TRUE	FALSE	435.78	620.71	669.02	807.22	871.38
4	FALSE	FALSE	TRUE	TRUE	510.42	700.14	760.27	903.14	957.03
5	FALSE	TRUE	FALSE	FALSE	480.17	668.34	717.18	854.73	888.22
6	FALSE	TRUE	FALSE	TRUE	640.16	868.72	901.33	1014.61	1029.79
7	FALSE	TRUE	TRUE	FALSE	430.54	604.41	642.33	760.70	795.80
8	FALSE	TRUE	TRUE	TRUE	505.73	687.78	740.64	852.68	888.14
9	TRUE	FALSE	FALSE	FALSE	466.05	631.62	644.53	748.28	806.69
10	TRUE	FALSE	FALSE	TRUE	635.31	854.01	858.07	890.26	881.64
11	TRUE	FALSE	TRUE	FALSE	391.81	552.63	595.40	733.48	796.86
12	TRUE	FALSE	TRUE	TRUE	474.12	626.40	666.25	805.22	867.93
13	TRUE	TRUE	FALSE	FALSE	465.17	629.98	637.26	721.83	744.11
14	TRUE	TRUE	FALSE	TRUE	634.34	852.86	858.08	890.26	851.14
15	TRUE	TRUE	TRUE	FALSE	386.82	535.37	566.49	683.08	717.76
16	TRUE	TRUE	TRUE	TRUE	469.86	611.95	640.39	755.17	787.45
				Average:	503.09	686.97	721.06	832.88	866.14

Consistently, the best performing machine line without a freezer (FALSE) is 15, and the best performing machine line with a freezer (TRUE) is 16, across all production plan sizes. It should come as no surprise that these lines are the most advanced in the experiment, and have all the other advanced machine variants enabled.

With the use of this data we can calculate the effect e of enabling the different machines, using simple ANOVA 1-factorial experiments. This is done by comparing the solution with the machine, $S_{True,i}$ to the solution with the same system settings but without the machine, $S_{False,i}$. Since there are 16 solutions, and thus 8 pairs of True and False solutions for every machine, we use the formula:

$$e_1 = \frac{1}{8} * \sum_{i=1}^{8} (S_{True,i} - S_{False,i})$$

to find the average added value. The added value of an average line with one of the advanced machine variants enabled vs. not enabled is shown in Table 13 for experiment run 1, and Table 14 for experiment run 2. This represents the extremes of the curve: if it were possible to turn on *half* a machine, this would result in obtaining the average result. Note that the these numbers are negative for all machines, except for the freezer. This is because the freezer adds much extra time to the cooling step, whilst the other machines make the process a lot more flexible and thus help to save time. Since our objective is to minimize the makespan, negative numbers represent good results in

these tables. Based on this table, if a hypothetical bakery had a production plan with 15 recipes, and only enough money to upgrade one of these machines, we would recommend they pick the advanced Climate Closet, since that will save them an average of 108.9 minutes in makespan.

Table 13: Impact of machines on makespan in minutes (left) and as a percentage of total makespan (right), based on experiment 1.

#Recipes	Climas	Oven_P	D_cooler	Freezer	Climas	Oven_P	D_cooler	Freezer
3	-25.3	-3.0	-104.9	121.6	-5.0%	-0.6%	-20.9%	24.2%
7	-50.2	-9.1	-139.1	144.5	-7.3%	-1.3%	-20.2%	21.0%
10	-75.5	-16.2	-121.9	141.1	-10.5%	-2.2%	-16.9%	19.6%
15	-108.9	-32.5	-90.6	119.3	-13.1%	-3.9%	-10.9%	14.3%
20	-118.9	-56.7	-61.7	98.0	-13.7%	-6.5%	-7.1%	11.3%

Table 14: Impact of machines on makespan in minutes (left) and as a percentage of total makespan (right) based on experiment 2

#Recipes	Climas	Oven_P	D_Cooler	Freezer	Climas	Oven_P	D_Cooler	Freezer
10 S	-78.5	-14.8	-116.1	147.6	-12.9%	-2.4%	-19.1%	24.3%
10 M	-71.2	-14.7	-125.8	143.9	-9.8%	-2.0%	-17.4%	19.9%
10 L	-114.1	-61.6	-89.3	181.9	-10.9%	-5.9%	-8.5%	17.3%

From these results we can deduce some basic conclusions about the effects of the machines on average makespan. The oven partition step only has limited use in instances of small to medium batch sizes, probably because potential waiting time before the oven is outweighed by waiting time in other, more influential steps like the cooler. The double cooler becomes less effective in large batches, or if many recipes are in a production plan. It is clear that the freezer adds significantly to the overall processing time, although the impact of this effect decreases when the number of batches increases. Overall, it should come as no surprise that lower average makespan is achieved in each case if the machines use the advanced variant, and that adding a freezer to the line increases the makespan.

However, we should not forget about the interaction effect between different machines. The added effects of two machines can help offset the added time cost of, for example, a freezer step. We alter the formula to compare two factors at the same time, with four pairs with and without both factors:

$$e_{1,2} = \frac{1}{8} * \left(\sum_{i=1}^{4} \left(S_{True,True,i} - S_{False,True,i} \right) - \sum_{j=1}^{4} \left(S_{True,False,j} - S_{False,False,j} \right) \right)$$

Leading to the following tables with results, with Table 15 displaying the interaction effects per possible second-degree interaction. In Table 16 the same values are displayed as a fraction of the average makespan for this step for clarity. The interaction effects for experiment 2 are shown in Table 17.

Table 15: Average Interaction Effects between machine variants in minutes, experiment 1

Interaction Effect	3	7	10	15	20
Climas + Oven_P	0.21	0.46	0.68	0.77	-6.49
Climas + D_cooler	-14.65	-21.44	-10.44	22.17	33.29
Climas + Freezer	4.33	4.43	3.64	-5.78	-17.33
Oven_P + D_cooler	-1.81	-6.00	-9.09	-16.87	-19.34
Oven_P + Freezer	0.23	1.33	3.15	3.76	4.65
D_cooler + Freezer	-42.82	-66.19	-57.57	-36.40	-18.32

Table 16: Average Interaction Effects between machine variants as a percentage of average makespan, experiment 1

Interaction effect	Total	3	7	10	15	20
Climas + Oven_P	-0.1%	0.0%	0.1%	0.1%	0.1%	-0.7%
Climas + D_cooler	-0.2%	-2.9%	-3.1%	-1.4%	2.7%	3.8%
Climas + Freezer	-0.1%	0.9%	0.6%	0.5%	-0.7%	-2.0%
Oven_P + D_cooler	-1.4%	-0.4%	-0.9%	-1.3%	-2.0%	-2.2%
Oven_P + Freezer	0.3%	0.0%	0.2%	0.4%	0.5%	0.5%
D_cooler + Freezer	-6.5%	-8.5%	-9.6%	-8.0%	-4.4%	-2.1%

Table 17: Interaction Effects between machine variants, in absolute minutes and relative percentages, experiment 2

Interaction effect	10S	10M	10L	10S	10M	10L
Climas + Oven_P	-0.51	0.96	-40.29	-0.1%	0.1%	-3.8%
Climas + D_cooler	-15.44	-10.58	40.88	-2.5%	-1.5%	3.9%
Climas + Freezer	-0.44	3.05	49.81	-0.1%	0.4%	4.8%
Oven_P + D_cooler	-7.52	-8.27	32.90	-1.2%	-1.1%	3.1%
Oven_P + Freezer	1.68	2.36	46.00	0.3%	0.3%	4.4%
D_cooler + Freezer	-55.73	-63.40	-109.83	-9.2%	-8.8%	-10.5%

From these two experiments it is clear that the most impactful combination of machines, if they are both enabled, is the combination of a double cooler and freezer. This makes sense: adding a freezer to the line vastly limits the flexibility of the cooling spiral whilst also lengthening the makespan of any product on the line. Adding a double cooling spiral will decrease the negative effects this has, especially if the batches are larger on average and less opportunities exist to match recipes of the same speed. As such, adding a double cooler to a line where freezing the buns is a necessary part of the production step should be realistically considered. Another interesting combination would be the advanced variant of the rising chamber (Climas) with the double cooler. That would be especially helpful if the number of recipes in a production plan is smaller than ten, or with small batch sizes, in this example. If batch sizes are large, combining the climate chamber with oven-partition has a large impact.

6.5) Conclusion

Using our model, we provided clear answers to a number of customer questions regarding scheduling dilemmas that Kaak engineers currently would have spent weeks debating. We also tested our solution methods, resulting in a belief that for big experiment runs, using Simulated Annealing methods is faster than Tabu search. However, for singular customer questions, there is no reason not to also utilize the Tabu-search functions, since the method only takes moderately longer than Simulated Annealing on big production plans, and may come up with better answers. We found a good overview of the singular- and interaction-effects of the different machine models, which will provide Kaak with a lot of insight into the added value of potential new machine arrangements, and the added benefit of flexibility on the line. Adding flexibility generally seems to improve the use of available production capacity. The results in this chapter are interesting, but come with a small caveat: they are only tested on a single dataset of bread recipes from a single customer. Whilst the diversity in this dataset is quite big, results may vary from customer to customer.

Chapter 7: Conclusions & Recommendations

7.1) Conclusions

In this thesis we researched the Industrial Bakery Scheduling problem, and came up with a novel calculation model to handle the continuous, batch-driven nature of the system. The model seems to perform well and serves as a good proof of concept for the problem Kaak is facing. The calculation model holds up to scrutiny and can be extended to handle many more machines and recipes if necessary. It is efficient at calculating the required minimal postponement times. We believe the model could also be applied to other scheduling problems facing the agri-food and/or manufacturing industry, where different batches of thousands of identical products are produced on the same continuous production line, with sequence-dependent change-over times in between batches. Think for example of a beer bottling plant, or a pasta factory.

Our optimization methods were based on tried and tested techniques, which performed as well as we could have expected. We did not find any unexpected results from utilizing the optimization techniques we did in our model. For the size of the problem, the current optimization techniques perform well enough, but further methods could be the subject of future research. Also, our “Hybrid” flow-shop model only had a single stage at which multiple machines could be chosen. In a model with multiple stages with multiple machines, the solutions provided by our model may become less deterministic and harder to solve. If such a case were to occur, increasing the range of possible optimization methods may be necessary. That is not very likely to happen for the small-bread line, but a problem Kaak could encounter is when two near-identical lines are placed in the same factory, and a choice has to be made which recipes to produce on which line. This would also be a situation in which the research by Jordan & Graves (1995) would be more applicable.

In initial experiments on an example customer support case, we used the model and the optimization methods to provide good answers to the questions that were posed. We found product sequences that were optimal, alternative production plans that lock a certain recipe in a sequence, and clear answers to how to balance production plans to accommodate new recipes. This same method of working can be extended to further customer cases without much issue. Furthermore we experimented with the effects of production plan- and batch-size on makespan, in combination with 16 different versions of the machine line, leading to new data regarding the performance of certain machine lines and machine combinations. This serves as a good showcase of the flexibility of the model to encompass different machine models, and the relative ease with which new machines can be added by altering the formulas for that specific machine.

7.2) Recommendations

The use of the tool that was built will help Kaak and its customers to make better use of time on new and existing machine lines, by optimizing the usage of change-over time between different batches. A simple check before recommending a production plan could save a great deal of time, in essence adding production capacity, and thus value, to production facilities. The tool will allow Kaak to properly advise its customers about the upgrading or integration of newer, more flexible machine models into the production process. It can also serve as a guide to the added value of (further) developing those more flexible machine models. We would advise to continue development of this tool, for example by extending the range of machine variants in the model. It should be possible to apply the same model to the loaf-bread line without much trouble, since it shares many similarities with the small-bread line. Building a proper user interface may help to increase the value to staff in the sales and engineering departments.

In terms of further research relating to the Flow Shop model, we would advise Kaak to perform an extensive survey to check the types of bread that its' many customers produce, and to create a complete dataset per customer with the production-times of recipes concerning different production steps. Especially the pre-rise step deserves more attention in our opinion, since we had to skip that step in our Flow-Shop model due to lack of data. The scheduling of recipes in the pre-rise step could prove to be of great influence on the outcomes of our model. Looking into change-over times regarding different types of product carriers would be another factor that has to be considered in an extended version of the model: whilst we assumed that all small-bread lines use a unified model of peel boards, some exceptions exist that may necessitate a more impactful change-over between different bread types. Also, differences in tact-time may have a great impact on the types of change-over that are allowed, which was a factor that is lacking in this thesis.

Kaak has struggled with simulating their complete machine lines for a few years by this point, and they initially reached out to the University of Twente for help to develop a Discrete Event Simulation (DES) model of their small-bread line. We stopped development of that model due to the scheduling problem that is the subject of this thesis and a lack of verifiable data on the stochastic aspects of the production process. Now that the Industrial Bakery Scheduling problem has been solved, we advise they look into building that model again. Especially machines that are prone to jamming, such as the packaging lines, as well as machines that require lots of physical intervention, such as the loading of dough into the dough dispenser, or the part of the line between the rising chamber and oven, would be well-served by building a DES model to help test the robustness of the different production plans that will be recommended by our flow-shop solution. Specifically, the movement of personnel to make physical alterations to the machines as a part of either change-over processes or due to breakdowns cannot be simulated in our flow-shop model, since these problems are discrete in nature and happen with a lot of uncertainty. A more robust DES solution is needed here.

Another, much more technical, problem Kaak regularly encounters is the simulation of their Programmable Logic Controllers (PLC's), the electrical components that send complex movement orders to the machines they build. Simulating a single machine is relatively easy, but since machines always work together in a complex line this presents a huge problem. These problems are too large and intensive to solve in their regular tools, such as NX-MCD. That goes for the manpower that is needed to program the problems, as well as the computational limits of their PCs. For this, looking into a DES-PLC programming solution may be key, since it should be possible to leave out many of the mechatronic aspects of the PLC simulation. This is truly out of scope for this project, however.

List of common phrases and abbreviations

Phrase	Description
DES	Discrete Event Simulation
(Product) Carrier	Depending on the factory, dough balls will be dispensed onto or into a product carrier in a set pattern, to protect their shape and prevent stretching. In the small-bread line, so called “peel boards” are the most common. Other product carriers include baking pans, in which loafs of bread are baked with a lid on top, or wave-carriers which are sometimes used as heat-resistant open baking pans for baguettes. We calculate production throughput in terms of Product Carriers, even though they are removed halfway through the process.
Climas/ Rising Chambers	Climate Chambers, place where dough rests and rises during production run. The temperatures and humidity are kept at a constant level. Commonly called Rising Chambers.
Flow-Shop	Production environment where all products follow the same predictable route along different machines.
HFP	Hybrid Flow-shop Problem, a deterministic problem where products follow a set of steps through different processing stations, and where at least one step has at least two independent stations.
Makespan	Total production time, from first entry onto the line to final exit from final machine, of individual batch of products or entire production plan, depending on context.
NEH	Abbreviation of “Newaz-Enscore-Ham”, the authors of the paper in which they first described their efficient cheapest-insertion construction heuristic applied to flow-shop models
PLBs	Peel Boards. See “(Product) Carrier”.
PLC	Programmable Logic Controllers: the computers that send machines their commands, to direct movements or responses to sensor- or user-input.
Postponement Time	The time that is added as waiting time *before* a recipe is first dispensed onto the production line, to avoid waiting times during the continuous production run.
Production Plan	A set one or more of recipes and the amount in which they will be baked on a given production day, in a specific sequence.
Recipe	A specific type of bread and the processing times, tact-time, baking temperature and other factors associated with producing it.
SA	Abbreviation of Simulated Annealing, an improvement heuristic in which random neighboring solutions are chosen, and sometimes accepted based on a certain acceptance-factor
Sequence	The sequence of different recipes in the production plan, the order in which they will be produced.
Tabu (search)	An improvement heuristic in which a “tabu”-list of previously-visited solutions is kept. The solutions on this list are forbidden to re-visit, leading to a diversification of the algorithm.
Waiting Time	Any time a product has to wait to enter a machine, either due to it being occupied or changing over to new settings. In our model, waiting time is not allowed and instead leads directly to Postponement time.

Bibliography

Allahverdi, A., & Aldowaisan, T. (2001). Minimizing Total Completion Time in a No-Wait Flowshop with Sequence-Dependent Additive Changeover Times. *The Journal of the Operational Research Society, 52*(4), 449-462.

Allahverdi, A., Ng, C. T., Cheng, T. C. E., & Kovalyov, M. Y. (2006). A survey of scheduling problems with setup times or cost. *European Journal of Operational Research, 187*, 985-1032.

Ben-Daya, M., & Al-Fawzan, M. (1998). A tabu search approach for the flow shop scheduling problem. *European Journal of Operational Research, 109*(1), 88-95. doi:10.1016/S0377-2217(97)00136-7

Graham, R. L. L., E.L.; Lenstra, J.K.; Rinnooy Kan, A.H.G.,. (1979). Optimization and approximation in deterministic sequencing and scheduling: a Survey. *Annals of Discrete Mathematics, 5*(287-326).

Gupta, J. N. D. (1988). Two-Stage, Hybrid Flowshop Scheduling Problem. *Journal of the Operational Research Society, 39*(4), 359-364.

Hall, N. G., Laporte, G., Selvarajah, E., & Sriskandarajah, C. (2003). Scheduling and Lot Streaming in Flowshops with no-wait in process. *Journal of Scheduling, 6*, 339-354.

Jordan, W. C., & Graves, S. C. (1995). Principles on the Benefits of Manufacturing Process Flexibility. *Management Science, 41*, 577-594.

Kirkpatrick, S., C. D. Gelatt, J., & Vecchi, M. P. (1983). Optimization by Simulated Annealing. *Science, 220*, 671-680.

Lee, C.-Y., & Vairaktarakis, G. L. (1994). Minimizing makespan in hybrid flowshops. *Operations Research Letters, 16*(3), 149-158.

Nawaz, M., Enscore, E. E., & Ham, I. (1982). A Heuristic Algorithm for the m-Machine, n-Job Flow-shop Sequencing Problem. *The International Journal of Management Science, 1a*(1), 91-95.

Pan, Q.-K. (2015). An effective co-evolutionary artificial bee colony algorithm for steelmaking - continuous casting scheduling. *European Journal of Operational Research, 250*, 702-714.

Reisman, A., Kumar, A., & Motwani, J. (1997). Flowshop Scheduling/Sequencing Research: A Statistical Review of the Literature, 1952-1994. *IEEE Transactions on Engineering Management, 44*(3), 316-329.

Ruiz, R., & Maroto, C. (2004). A comprehensive review and evaluation of permutation flowshop heuristics. *European Journal of Operational Research, 165*, 479-494.

Ruiz, R., Şerifoğlu, F. S., & Urlings, T. (2008). Modeling Realistic Hybrid Flexible Flowshop Scheduling Problems. *Computers & Operations Research, 35*(4), 1151-1175.

Taillard, E. (1990). Some efficient heuristic methods for the flow shop sequencing problem. *European Journal of Operational Research, 47*, 65-74.