MASTERARBEIT / MASTER'S THESIS

Titel der Masterarbeit / Titel of The Master's Thesis

"Smart manufacturing – a simulation-based evaluation of different sequencing rules for a stochastic flexible flow shop"

verfasst von / submitted by

Zsófia Kunczi (BA)

angestrebter akademischer Grad / in partial fulfillment of the requirements for the degree of

Master of Science (MSc)

Wien, 2018 / Vienna, 2018

Studienkennzahl lt. Studienblatt / degree programme as it appears on the student record sheet:	A 066 914
Studienrichtung lt. Studienblatt / Degree programme as it appears on the student record sheet:	Masterstudium Internationale Betriebswirtschaft / Master of International Business Administration
Betreut von / Supervisor:	Univ.-Prof. Dr. Karl F. Dörner
Mitbetreut von / Co-Supervisor:	DI Dr. Roland Braune

Abstract

The following master thesis deals with a problem in connection with a field of study that is gaining more and more attention, since the automation of work processes is becoming reality not only in production, but in all areas of our lives. The fourth industrial revolution, as it is called in manufacturing, refers to intelligent systems that are able to communicate and cooperate dynamically with each other and therefore either facilitate or even take over decision-making. The simulation model developed for and described in this thesis is the implementation of a production process of a Flexible Flow Shop type system, which is not only capable of providing answers to various flow shop problems, but also of testing solutions and thus optimizing complex processes. Five different scenarios were tested and analysed to demonstrate the effectiveness of the model. This work is only a small demonstration of what this model can be applied for, as it has been designed with a high degree of flexibility and adaptability and taking into account possible future research.

Deutschsprachige Zusammenfassung

Die folgende Masterarbeit beschäftigt sich mit einem Problem im Zusammenhang mit einem immer mehr an Bedeutung gewinnenden Fachgebiet, da die Automatisierung von Arbeitsprozessen nicht nur in der Produktion, sondern in allen Bereichen unseres Lebens Realität wird. Die vierte industrielle Revolution, wie sie in der Fertigung genannt wird, bezieht sich auf intelligente Systeme, die in der Lage sind, dynamisch miteinander zu kommunizieren und zu kooperieren und somit Entscheidungen zu erleichtern oder gar zu übernehmen. Das für diese Arbeit entwickelte und beschriebene Simulationsmodell ist die Implementierung eines Produktionsprozesses eines flexiblen Flow-Shop-Systems, das nicht nur in der Lage ist, Antworten auf verschiedene Flow-Shop-Probleme zu geben, sondern auch Lösungen zu testen und damit komplexe Prozesse zu optimieren. Fünf verschiedene Szenarien wurden getestet und analysiert, um die Wirksamkeit des Modells zu demonstrieren. Diese Arbeit ist nur eine kleine Demonstration dessen, was man mit diesem Modell erreichen kann, da es mit einem hohen Maß an Flexibilität und Anpassungsfähigkeit und unter Berücksichtigung möglicher zukünftiger Forschungen entwickelt wurde.

Acknowledgement

I would like to mention some very important contributions to this work. First of all, I would first like to thank the department of Production and Operations Management with International Focus and especially Prof Karl Dörner for giving me the opportunity to write my thesis under his supervision. In addition, I would like to thank him for his input and support throughout my writing of the thesis.

Second of all, I would like to thank my co-supervisor DI Dr Roland Braune, who has introduced me to this very interesting and challenging topic and supported me throughout my master thesis and whom I thank not only for his technical support, but also for his time and patience.

Finally, I would like to thank my family, who have always been at my side not only during my master thesis but my entire studies, and moreover thank my boyfriend for his great support in specific programming questions which was so crucial for this very work and for always being there for me in everyday life.

Contents

List of abbreviations

HFS	Hybrid Flow Shop
FFS	Flexible Flow Shop
FMS	Flexible Manufacturing System
B&B	Branch & Bound
BS	Buffer Slot
TS	Transport Slot
PR	Priority Rule
C_{max}	Makespan or maximum completion time
PT	Processing Time
ST	Setup Time
TT	Transport Time
WT	Waiting Time
SQ	Shortest Queue
SPT	Shortest Processing Time
SST	Shortest Setup Time
STT	Shortest Transport Time
SWT	Shortest Waiting Time
M0-M8	Machine 0 – Machine 8
S1-S5	Scenario 1 – Scenario 5

List of figures

List of tables

1. Introduction

1.1. Related Work

A lot of research has been conducted that relates to problems in flow shop production, as there are many different aspects to consider that can change the outcomes of a flow shop schedule. We would like to return to the most important ones in the section 3 under *Literature Review* and continue with the current background of this topic: a project that arose from a cooperation between the Faculty of Production and Supply Chain Management of the University of Vienna and "Syngroup Vienna", a management consultancy which focuses on Industry 4.0 applications. In the course of this cooperation, machine learning was proposed in order to generate a general priority rule in the form of a decision tree for scheduling the different types of jobs and to improve the autonomous planning system of the industry partner. In order to demonstrate the effectiveness of the approach, various dispatching rules for global performance indicators, such as total lead time, were tested (Benda, Braune, Doerner, & Hartl, 2018).

In this research we will draw on the starting point and process structure of the aforementioned paper and proceed in a similar way but using a completely different implementation approach. The outcomes of different sequencing rules in specific scenarios are provided using a graphical programming tool, AnyLogic, which we used to create a simulation to derive a possible pattern for solving similar problems in production. Another very important difference that needs to be highlighted is the dynamic environment of the model presented here. We have incorporated some stochastic elements into the model that bring the problem even closer to reality.

1.2. Subject of the Thesis

The underlying thesis involves the investigation of a real-world motivated problem by a simulation model which evaluates the effects of a number of sequencing rules that can be applied for different types of orders as well as for different machines of a stochastic and flexible flow shop system. The simulation provides the system with a specific logic in each displayed rule. These rules are going to be compared across different simulation scenarios and runs and on the basis of this comparison we would like to present the best priority control

solutions projected on different goals (e.g.: Minimization of the overall margin / Maximization of machine utilization / Minimization of setup times). The best rules are tested not only in deterministic but also in stochastic cases. As stochastic cases we understand fluctuating processing times. The biggest challenge by far is to develop a simulation logic that is flexible enough to test different setups without changing every little detail of the simulation. The overall goal is to find the best dispatching rules with the main objective of the minimization of total completion time (=makespan) in the presented scenarios.

1.3. Motivation – Industry 4.0

The motivation for this work stems on the one hand from a general interest in production processes and procedures, in particular in "intelligent solutions" in manufacturing, also known as the fourth industrial revolution, and on the other hand from a preference for graphical programming and practical approaches. AnyLogic Software offers extensive possibilities for both. It not only has the ability to simulate different scenarios and experiment with new solution logics, but also an enormous potential for solving optimization problems. Besides, to the best of our knowledge, as it will be highlighted in the literature review section below, the specific combination of constraints in connection with the flow shop system that the thesis further investigates in a heuristic approach using simulation experiments has never before occurred in scientific sources (Benda, Braune, Doerner, & Hartl, 2018). Ruiz & Vázquez-Rodríguez (2010) argue in their concluding words that it is very unlikely that a real-world problem exactly corresponds to any of the models studied in the literature. We take the other path in the sense that we copy a real-world problem and implement it in a simulation as accurate as possible also implementing stochastic variables that can represent real-word unpredictability. They state as well, that:

> *"it seems to be a more promising strategy to generate heuristics which show flexibility on a wide range of HFS problems. It is also important to consider that the real world is unpredictable and dynamic. Algorithms must be able to find solutions which remain robust under different scenarios. No results have been found on robust scheduling in HFS"* (Ruiz & Vázquez-Rodríguez, 2010, S. 14).

Therefore, we are motivated to find a meaningful simulation environment to test many real-world, dynamic problems while remaining robust and reliable in terms of performance.

2. Problem Statement

2.1. Description of the underlying problem

Our problem differs in several ways from a "basic" flexible flow shop (FFS) described in section 3.1. The underlying problem is specified using the following figure (*Figure 1*) which is an abstract model of the FFS configuration borrowed from the resulting paper of the collaboration between our faculty and Syngroup Business Consultancy.

Figure 1 illustrates the starting point for this master thesis by the case study on which Benda et al. have worked on with the aforementioned consultancy and which we intended to expand and examine in this thesis from a new perspective.

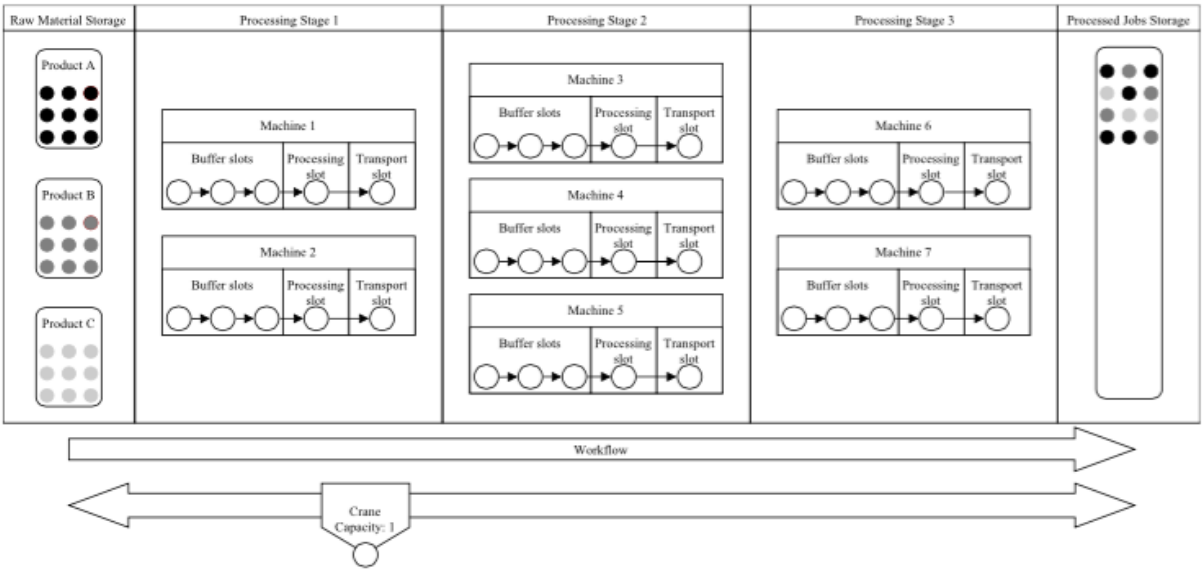

Figure 1: Shop floor configuration (Benda, et al., 2018, p. 2)

This illustration shows similarities to the one described in the literature review (*Figure 2*), but on closer inspection, we will notice that many constraints have been added to the basic problem. Nonetheless, before proceeding with the limitations, it should be briefly addressed what is referred to as the "basic problem". In essence, Baker (1975) describes the flow shop problem as a problem that *"involves processing n jobs on m machines where the flow of work is unidirectional"*. On the contrary, when we use the word flexible flow shop, routing flexibility becomes an essential factor. In other words, this means that instead of a straightforward direction for the workflow, *"alternative paths can be followed through the system for a given process plan"*. In this way, the classic formulation is extended to an

FFS/HFS/FMS problem (Rossi & Dini, 2007, S. 503). The underlying study may in principle be described as an "all in one" problem, since almost all kinds of limitations can be found in one setup. It has three different types of resources (Product A/B/C), sequence dependent setup times and even blocking constraints (limited buffers before and after the "Processing slots"). The orders must go through all processing steps, processing steps must not be skipped. Neither is interruption during production allowed (=no preemption). Although these are all commonly encountered characteristics and can be found separately in different applications, they rarely occur simultaneously. According to a very recent study of our faculty members, the discussion of the combination of these properties has never been published before (Benda, Braune, Doerner, & Hartl, 2018). Moreover, we must also take into account, that the orders may have different processing times on different machines and that in this problem realization we only have one transport unit that transports the orders from stage to stage and machine to machine. Additionally, this transport unit will travel longer from the first machine to the third machine than e.g. from the first machine to the last machine and vice versa.

If we were to examine the differences between the basic FFS and our problem, the first thing we could see is, that instead of simply having a number of "unfinished products" on the Raw Material Stage as on *Figure 2*, we have three different types of raw materials waiting to be processed on *Figure 1*. These types of products are considered as alternative resources and are differentiated by different colours both in the related work of Benda et al. and our implementation in AnyLogic later on. Product type A is marked with Red, B with Blue and C with Yellow. Although these types differ in some respects and complicate the problem, they all can be processed on every machine and transported by the same transport unit at the same speed. We also have the same initial amount of all job types in the simulation model and manufacturing is not terminated until all of them have left the system. It is not necessary for any of the jobs to be finished before another job can be started.

Depending on the type of order, we have machines that can process some types faster than others. We could say, that the machines are somewhat different from each other, or that it is simply due to the fact that specific machines are better suited for one type of job than for others. This characteristic arises from the fact that in this problem we are dealing with so-called "unrelated parallel machines" which means that *"the processing times of a job in a stage depend on each specific machine within this stage"* (Ribas, Leisten, & Framin, 2010, S.

1441). Another important difference is in the time needed to set up a machine whenever a new type of order arrives at this machine. This time is called setup time, which we will elaborate on in section 3.1.2, but the point here is, that setup times can only occur if there are alternative resources.

To sum up, this means that the different types are not completely unrelated, as they all can be processed by the same machines and in fact all have to go through the same stages. However, they must be "treated" differently in the simulation, meaning that some of them will take longer to process than others and that each time job types alternate on the machines they will trigger a setup time.

We will return to the special features and the differences between this model and the configuration in the AnyLogic model in the following chapters. However, the most striking characteristics can already be declared, such as the fact that in our case a factory with nine machines is being investigated with three parallel machines at each stage and that the simulation comprises 1000 jobs per job type, with a number of job types remaining the same.

Some other words on our objectives: Our goal is to demonstrate the effect of different sequencing rules on the output using simulation. When it comes to process optimization, this goal usually involves makespan minimization, which is the minimization of the completion time of the last job. Minimizing mean flow time, which is defined as average job completion time, is another common objective of optimization procedures (Quadt & Kuhn, 2007, S. 687). Since practical applications show the importance of setups, we would like to include the number of setups into our evaluation tables in the *Results* section as well as the mean utilization rate of the machines as further evaluation criterion.

As has already been emphasized, in regard to our objectives, setup times are very important, due to the fact that by minimizing the setup times, we are able to minimize the maximum completion time. Setup times occur every time we change the type of jobs independently from machines. In our general setup, these times can be illustrated as a 3x3 matrix:

	Red	Blue	Yellow
Red	0	12	10
Blue	12	0	8
Yellow	10	8	0

Table 1: Setup times (short) between different types of jobs

Every time a job marked blue follows a red-job on a machine, we have to consider 12 additional time units besides the processing time as a total delay time caused by the particular machine. In case a yellow-job follows a red one, we consider 10 time units and so on. As our matrix is symmetrical, these times apply vice versa. However, this setup time matrix should not necessarily be symmetrical, and it usually is not. In most cases, one models setup times by means of asymmetric setup matrices, since it cannot be assumed that the time required for setting up a machine is the same if, for example, red follows blue and blue follows red.

Another significant problem component is the difference in terms of storage space between the standard FFS problem and our FFS realization. *Figure 1* elucidates that the buffer before and after the stages has decomposed into a so-called *buffer slot* for no more than two orders and a *transport slot* with a capacity of one. These capacities will be changed to three in case of the buffer slot and two in case of the transport slot in the simulation, but the principle remains the same. In our problem, orders can remain on a stage after completion and block the production flow until a buffer space becomes available for them on the next stage. Since this blockage can very easily break down the simulation we had to develop a congestion prevention tool already at an early stage of the programming work. This means that whenever the capacity of a “transport slot” (the storage space behind the machines) is exhausted, the first item in this small queue gets the highest priority and must be transported to the next level to prevent blocking. The way how we can track this information is described in *Implementation* section.

3. Literature Review

3.1. Flexible Flow Shop Scheduling

As already mentioned, there is a vast amount of research related to the theory of flow shop planning and sequencing rules. The problem can be viewed from very different angles. There are some that focus on a single machine and a large number of jobs, others that examine multiple machines and distinguish between heuristic, metaheuristic and optimal procedures, not to mention the distinction between deterministic and stochastic approaches. The problem that most accurately describes our case is referred to in the literature as a "hybrid flow shop" (HFS) or "flexible flow shop" (FFS). These kinds of problems can be found in a considerable number of industries and are most popular in the process industry. There are numerous examples in the literature, such as the chemical industry, electronics manufacturing, textile manufacturing and herbicides production, the food and cosmetics, the pharmaceutical sector, the automotive industry, glass container fabrication, wood-processing, as well as the semiconductor industry (Quadt & Kuhn, 2007, S. 686).

Although the difference between standard and flexible flow shops will be discussed later by means of an FFS scheme, I would like to highlight the most important difference here: If there is more than one machine at least at one processing stage and there are at least two stages, we speak of a flexible or hybrid flow shop problem (Ruiz & Vázquez-Rodríguez, 2010, S. 1). However, this "small" difference complicates the problem of the rather simple concept of a flow shop to such an extent that it is not possible to solve it in polynomial time. It is proven that once we deal with a flow line problem consisting of two stages, it is sufficient that one of these two stages has more than two machines, while the other still has only one, we speak of an NP-hard problem (Gupta, 1988). For its complexity and also its practical relevance, the FFS problem attracted a lot of attention from researchers.

Already at the very beginning of the *Handbook on scheduling: from theory to applications* (Blazewicz, 2007) we are confronted with the term "permutation schedule". In a permutation flow shop, each machine processes the jobs in the same order. According to Blazewicz et al. (2007), the great majority of the literature is limited to a specific case of flow lines, namely to permutation schedules. These schedules have been shown to lead us to optimal solutions when we are looking for a minimum makespan in any flow shop scheduling problem consisting of a maximum of three machines. However, this optimal result cannot be further

developed, as they have demonstrated by the example of a 2-job 4-machine problem. When solving to optimality in *flexible* flow shops, both Quadt and Kuhn (2007) and Ruiz et al. (2010) claim that the preferred procedures are based on Branch & Bound (B&B). Salvador (1973), who was one of the first to investigate the HFS problem, has even combined his Branch & Bound method with a permutation schedule and employed dynamic programming for instances of a small size. This schedule simplified the problem fundamentally because one "only" had to choose one sequence of jobs that was consistently used at all processing levels. However, it is important to note again, that non-permutation schedules may result in a better makespan (Ruiz & Vázquez-Rodríguez, 2010, S. 3).

Besides exact methods there are also heuristic and metaheuristic methods to solve FFS problems. When it comes to heuristic methods, the simplest types of heuristics are different types of (list) scheduling algorithms or dispatching rules (Ruiz & Vázquez-Rodríguez, 2010, S. 3). Simulations, such as the one presented here are used to analyse the performance of these planning algorithms in comparison to each other and to test them for a variety of problems. There are examples of simulations that not only test the effectiveness of the rules in different scenarios, but also their behaviour in static and dynamic hybrid flow systems (Kadipasaoglu, Xiang, & Khumawala, 1997).

Now that we clarified the different approaches to deal with flexible flow shop problems, it is crucial to understand what it is exactly, or at least to provide a picture of how the "standard" form of such a flow shop problem looks like.

Quadt and Kuhn (2007) demonstrate this type of flow shop using the following illustration in their research paper about the taxonomy of flexible flow shops:

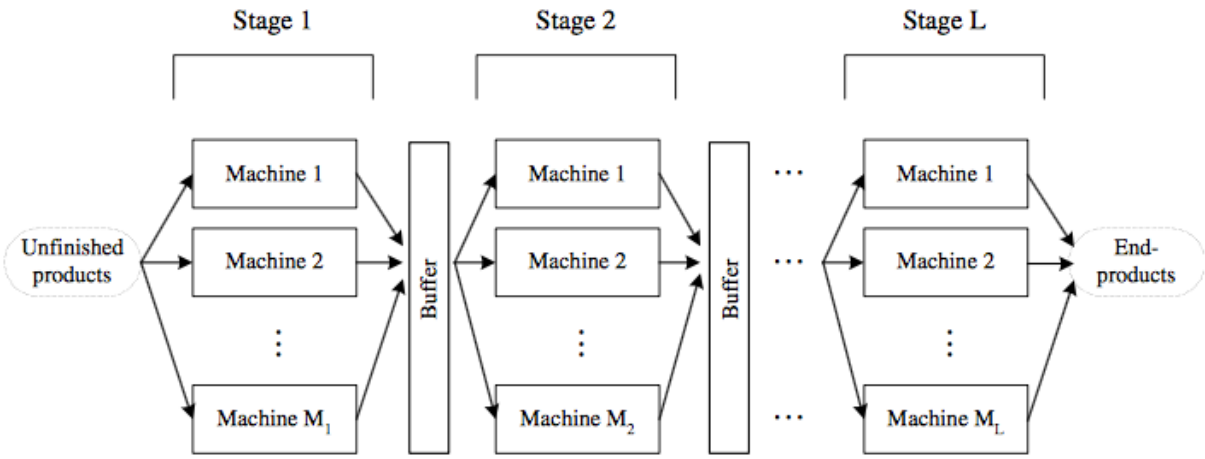

Figure 2: Schematic view from a flexible flow line (Quadt & Kuhn, 2007, p. 687)

How can we describe this special type of flow shop? As it is often the case, different sources describe the problem differently. It is important to understand that this type of flow shop also has different variants. The standard problem can be extended or limited with regard to real-life applications. Therefore, we have decided to use the above illustration from Quadt and Kuhn as a "template" for describing the standard from of the problem, just like the authors did.

As was mentioned earlier and as is elucidated in *Figure 2*, in an FFS problem there is more than one machine at least on one stage where the orders can be processed. Regardless of how many machines are on the stages, orders can only be processed by one of several machines at each stage. A further criterion is that the orders must be processed from the first to the last stage following the same production flow, while the machines can process a maximum of one order simultaneously and one order can be processed by one machine at the same time. In the basic version of the problem, all parameters are deterministic, and the parallel machines of each stage are identical. Preemption is not permitted, i.e. every time an operation has started on a machine, it must be finished without interruption. In addition, the "standard" form of the problem does not take setup times into account either. The input data is deterministic and known in advance and there is an unlimited buffer between all processing steps. Last, but not least, *"a job might skip any number of stages provided it is processed in at least one of them"* (Ruiz & Vázquez-Rodríguez, 2010, S. 1).

3.1.1. Sequence Dependent Setup Times

As far as setup times are concerned, only few studies explicitly consider operation/setup time between orders, which is – as already remarked – the time that is necessary to set up a machine before it can process a job. In our case, this is a constraint that is classified in the literature as a *"non-hybrid-specific job constraint"* since the setup times depend on the sequence and not on the machine (Ribas, Leisten, & Framin, 2010, S. 1442). This means that the time required does not depend on the assigned machine on a certain processing level, but on certain properties of the materials associated with two subsequent jobs, like their colour. According to Quadt & Kuhn (2007), most studies assume that no setups need to be performed at all or that setup times are sequence-independent and there is only one unit from each product type. In this case, they can argue that the setup time of a job can be added to its processing time and does not have to be taken into account explicitly. However, if, as in our case, setup times are sequence-dependent or if several jobs of the same product type have to be produced, setups must be explicitly taken into consideration.

Allahverdi et al. (2008) collected in their survey paper more than 300 papers that were published between 1999 and 2006. In this survey, they described their own work as a *"continuation of the earlier survey papers of Allahverdi et al. (1999) and Potts and Kovalyov (2000)"*, which had been reviewing literature on scheduling problems with separate setup times since the mid-1960s. The interest in planning problems that were related to setup times and costs had arisen since then (1999) and continued to increase, whereby more than 40 publications were appended to the literature on average per year (Allahverdi, et al., 2008, pp. 985-86). Using *Figure 3*, the authors classify research on scheduling problems with setup according to four important sub-categories: batch sequence dependent /- independent problems and non-batch sequence dependent /- independent problems.

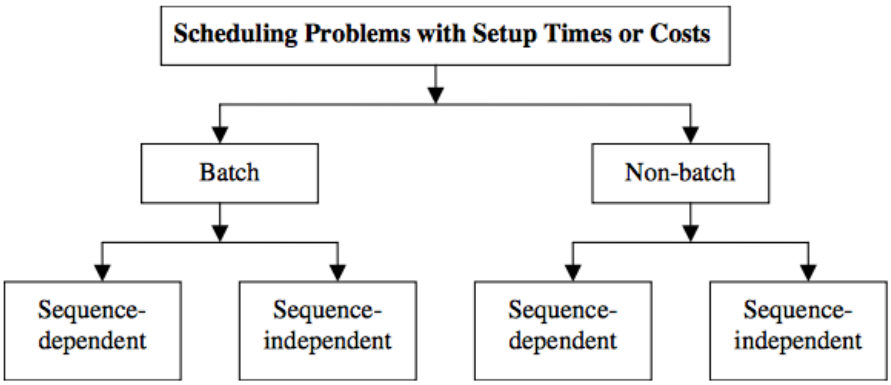

Figure 3: Classification of scheduling problems with setup times (Allahverdi, Ng, Cheng, & Kovalyov, 2008, S. 988)

While a batch is defined by its property that orders are divided into families and a set of orders of the same family is called a batch, a non-batch is a special case of batch where each family consists of exactly one job (Allahverdi, Ng, Cheng, & Kovalyov, 2008, S. 987). The underlying problem of this thesis would classify in our understanding as a parallel machine problem with unrelated machines and non-batch sequence-dependent setup times.

The big advantage of simulation is that even with one small change we can operate our model with any desired number and switch from asymmetric to symmetric matrices and vice versa any time. Furthermore, batches could also be implemented with little effort.

3.1.2. Blocking Constraints

The generic term “limited buffer” refers to a lack of storage space, which can be difficult to handle. Why do we have a “lack of storage space”? A buffer space between two consecutive stages / machines may have a limited capacity if the products are physically large (e.g., TV, copier). This can then lead to blockages. If a buffer is full and the preceding machine is not allowed to release a job into the buffer after processing has been completed, blocking occurs (Pinedo, Flow shops and flexible flow shops (Deterministic), 2016, S. 151).

There are several types of blocking constraints that have been examined and modelled in planning problems. Among others, Sawik T. (2000) has studied the problem of limited intermediate storage. In his publication, we can learn about a multi-stage hybrid flow shop system with identical parallel machines and an output buffer with limited capacity. In contrast to our example, his “output buffer” stores finished products instead of delivering them to the next stage for delivery to the customers. Akrami et al. (2006) investigate a system without buffers, but with a blockage resulting from zero processing times. Furthermore, there is the possibility, not only to limit the buffer locations for finished jobs, but to limit the storage time within a buffer such as in the case of perishable intermediate food products. Finally, there are some other publications that consider a “no wait limitation”, which means that the operations of a job must be processed from start to finish without interruption on or between machines (Ribas, Leisten, & Framin, 2010, S. 1441).

3.1.3. Stochastic FFS Scheduling

If we look for literature related to stochastic FFS systems, we will find that the selection of studies is not as large as in deterministic cases. However, events in real situations are often accompanied by uncertainties that may affect the decision-making process (González-Neira, Montoya-Torres, & Barrera, 2017). But what exactly is the difference between scheduling deterministic and stochastic flow shop systems? From a very practical point of view of the implementation process, there is not much difference between e.g. deterministic and stochastic processing times, yet they are very different from each other. In case of a stochastic flow shop problem, a few of the classic assumptions for deterministic flow shops must be replaced by some uncertainty factors. While in a deterministic case we anticipate that order release dates are known, and the machines are always available, in a stochastic case we are

assuming that the order release dates are not known in advance and that some machine failures may occur.

Although detailed examination of these differences would be certainly very interesting and could be included in our simulation with relatively little effort, there is currently no reliable data on machine failures or order generation for the case study under consideration. For this reason, we decided to focus on the third very important aspect that distinguishes deterministic and stochastic flow shops, namely the assumption that in a stochastic flow shop, – instead of independent and deterministic processing times – we cope with processing times defined by independent random variables. The state of the art in stochastic flow shop scheduling with random processing times is first described in an article by Gourgand et al. (2003), which even presents a classification of related work with regard to the number of machines (two or more), the buffer size (zero or unlimited), the distribution function of the processing times and finally the objective studied. This classification reveals that in most cases the target function investigated was $E(C_{max})$, which stands for the objective of finding a job schedule that can minimize the expected makespan. Another noteworthy characteristic about the presented classification is that in studies of more than two parallel machines in the system, both at zero and at unlimited buffer capacity, predominantly a general distribution function was applied for modelling the processing times. Only in two-machine stochastic flow shops with unlimited buffer capacity were other types of distribution functions such as exponential, geometric and Erlang used (Gourgand, Grangeon, & Norre, 2003, S. 417).

Even more remarkable, however, are the two methods utilized by the authors to solve the problem of performance evaluation. One can assume that evaluating the objective criterion is not easy when stochastic events disrupt the system. Nevertheless, the study proposes two approaches that help to overcome this challenge: *Markovian Analysis* and *Stochastic Discrete Event Simulation*. Since the assumptions in the first method do not correspond to our problem (e.g. unlimited buffer capacities), we would like to turn directly to the second method: Stochastic discrete event simulation. In the case of a simulation, we can carry out statistically independent simulations, so-called replications, several times in succession. For each replication, a certain sample of order processing times is selected, and the simulation terminates returning a certain minimum completion time for our criterion. Obviously, the number of replications must be large enough to ensure a good sample size to account for the

model's stochastic elements. After the replications are computed, the mean values of the features of interest (Minimum Makespan (C_{max}), Maximum Utilization (ρ_{max}) and Minimum Number of Setups) are estimates of the expected criteria (Gourgand, Grangeon, & Norre, 2003, S. 422-423).

Since this literature review presented by Gourgand, which has only investigated static FS problems under uncertainties, the field of planning problems has grown, and more complex applications can be found these days. In 2017, a new review on stochastic FS and FFS problems appeared, based on 100 publications between 2001 and 2016. Unsurprisingly the authors observed an increasing trend of scientific publications on FS and FFS under uncertainty factors since 2001. Similar to the deterministic case, there is still a significant difference between FS and FFS publication numbers, with FFS only playing a role in 25% of the revised work. Furthermore, we found it remarkable that stochastic processing times are most frequently studied of all parameters under uncertainty, which account for 79% of the analysed papers, while the second most common parameter is the machine breakdown (González-Neira, Montoya-Torres, & Barrera, 2017).

3.1.4. Dispatching Rules

Ouelhadj & Petrovic (2009) distinguish between different techniques such as heuristics, meta-heuristics, multi-agent systems, artificial intelligence techniques, and dispatching rules to solve dynamic scheduling problems. Their paper provides an overview of the state of the art in scheduling problems within dynamic contexts. With regard to dispatching rules, it has been argued that these rules play a significant role among the aforementioned methods. Throughout the years, many publications were issued that focus on the performance of several different scheduling rules under stochastic and dynamic conditions of job shops and flow shops. For this purpose, simulation was used, since it allows the execution of all kinds of rules and selects the one with the highest performance. The most important contributions in this area were conducted by Ramasesh (1990) and Rajendran & Holthaus (1999) with their comprehensive surveys of dispatching rules in dynamic manufacturing problems. Furthermore, Sabuncuoglu (1998) performed a simulation study on scheduling rules for flexible shop floor systems at different failure rates and changes in lead times. Finally, he concluded that *"no single rule is the best under all possible conditions"*. (Ouelhadj & Petrovic, 2009, S. 424-25).

4. Implementation in Anylogic

4.1. Structure & General Setup

Before we move to the general structure of the simulation model of the physical production line, it is essential to understand what a simulation is and what it is used for. According to the Oxford Dictionary, a simulation is *"a situation in which a particular set of conditions is created artificially in order to study or experience something that could exist in reality"* (Oxford Learner's Dictionary, 2018). A slightly different explanation can be found in the Cambridge Dictionary where simulation is described as *"a model of a set of problems or events that can be used to teach someone how to do something, or the process of making such a model"* (Cambridge Dictionary, 2018). We can agree, that both definitions describe the term very accurately and complement each other, since simulations involve *both* the process of making a simulation *and* the experimental use of the simulation in order to investigate a problem. After Pegden et al. (1995), the purpose of simulation is to describe the behaviour of systems, construct theories and hypotheses that explain the observed behaviour and finally use the model to predict future behaviour such as the effects of changes in the system or its functioning. Simulation has many great characteristics, e.g. that experiments with not yet existing systems or new adaptations to already existing systems can be analysed, the passage of time can be controlled and bottlenecks in material, production flow, etc. and other problems can be detected in silico. Another useful aspect of simulation model building is that other stochastic influences (as described in section 3.1.4) can then be conveniently included. An additional advantage is often the option to create graphical illustrations in a simulation tool and the possible embedding in heuristic optimization methods (simulation-based optimization).

For the sake of completeness, however, some disadvantages should be kept in mind: Model building requires special training, because the quality of the results strongly depend on the quality of the model and thus on the skills of the model builder. In addition, simulations can be time-consuming and expensive, and the results may not easily be interpreted (Pegden, Shannon, & Sadowski, 1995, S. 5-10). Yet, when we consider the problem described in the section 2.1, it becomes quite clear that the arguments in favour of using a simulation instead of a mathematical model prevail:

Firstly, solving a mathematical model that considers all the practically relevant properties of the real-world system (e.g. limited buffers, sequence dependent setup times and even random processing times for more than two stages and more than two parallel machines) is usually not possible in a reasonable time. Secondly, our project calls for a stochastic setting that would require a stochastic optimization approach (e.g. stochastic mixed-integer programming) which adds complexity. Finally, it is also worth pointing out that often management consultancies attach importance to interpretable and traceable decision criteria (such as priority rules).

Simulation models can be subclassified into a number of different dimensions (analog vs digital, stochastic vs deterministic, static vs dynamic, discrete vs continuous) (Pegden, Shannon, & Sadowski, 1995, S. 5-6). We categorize our simulation model as a dynamic stochastic discrete event simulation model. Dynamic means that the model changes its behaviour according to its own state variables, the term stochastic has already been explained as capturing random components of the system whereas randomness is unimportant for deterministic models. Finally, we also speak of a "discrete event", since the state variables of the system change only at separated points in time and not continuously. An example in our case would be the change of number of jobs in the buffer locations due to an event such as arrivals/departures or finalized production steps.

In the following, we will elaborate in detail on the way our production line structure has been implemented in AnyLogic, a graphically programmable simulation software we are using for this simulation model. The representation of the problem, which is illustrated in *Figure 4*, differs from what was discussed in connection with the related work of Benda et al. (*Figure 1*). Specifically, we confront a flexible flow shop with three stages and three parallel machines per stage, 9 machines in total. The machine designation starts with 0 (M0) at the top machine of stage 1 and ends at the bottom machine in the illustration on stage 3, which is labelled M8. In addition to the processing stages, we have a preliminary *Raw Material* stage with the three order types already presented, which are marked red, blue and yellow. The so-called buffer spaces in front of the actual processing are shortened with 'BS' and after processing with 'TS', where the latter stands for a transport slot. These buffers are simple *Queue* elements that can hold three jobs for 'BS' and two jobs for 'TS'. We also have the *Hold* elements from 'h0' to 'h8' which serve as "gates" for the orders stored in the transport slots and only open under a specific condition. This condition is fulfilled in the basic scenarios for only

one job with the highest priority at a time in the entire system. This is because we only have one transport unit, that can transport one job at a time. The highest priority is selected for every rule by means of a different characteristic or a set of characteristics, e.g. shortest processing time (SPT) on the next machine, shortest setup time (SST) on the next machine etc. or their combination. A more detailed description of these priority rules will be given in the next sections. As we are aiming to minimize the total expected completion time, we are naturally looking for the jobs with the lowest values specified by time, therefore the job with the minimum value must be set to 'true' whenever the *Hold* elements evaluate the condition.

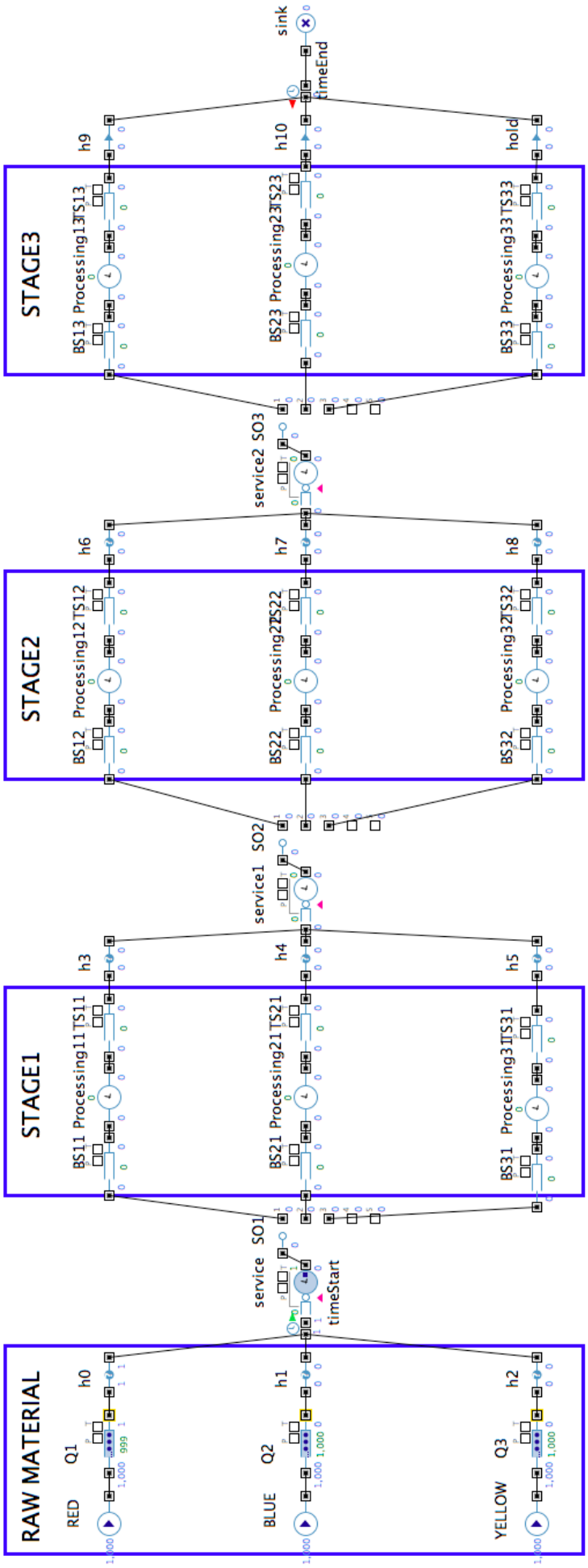

Figure 4: Representation of the problem in AnyLogic Software

Moreover, we have three *Service* elements which represent the transport between the respective machines. These elements are essential because the priority-status of each job will be updated when a service (=transport by *Crane*) is completed. For this update and to keep all important decisions synchronized over time, we have developed a function called 'update_policyRule'. This function is not only executed when a shipment is finished, but also when each order type enters the first "unlimited" buffer space of the raw material stage (Q1, Q2, Q3). This update decides first of all, which policy rule of the 16 possibilities applies to the job, then it decides which of the three parallel machines the respective jobs should ideally go to in the next step (function 'updateNextMachines' is called up) and finally it also decides on which of the available jobs should receive the highest priority and the label "true" in order to fulfil the condition required by the *Hold* element. You will be able to read more about these decisions under "Problem Decomposition".

To sum up what exactly happens during this implementation, we would like to present a simplified layout of the process. In this figure we see that the production flow is stopped each time by the hold elements. In order to control this process, as explained above, each job that is located on the first place of the initial queues (Q1,Q2,Q3) as well as of the transport slots (TS) will be evaluated using the "update_priorityPolicy" function. Only the order with the minimum time value will then be transferred by the *Crane* to the next stage and this continues until all jobs have been processed.

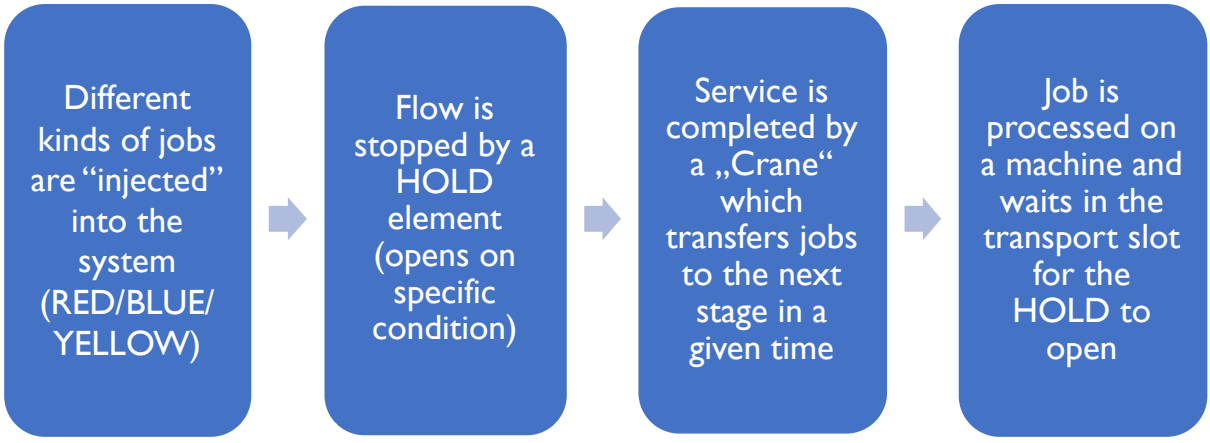

Figure 5: Simplified layout of the process

The most important parameters are introduced in more detail in the following section. Nevertheless, due to the complexity of the simulation model, a detailed description of all its components goes beyond the scope of this thesis.

4.1.1. Processing Times

The processing time is the time, that a single machine takes to process a single job. As we already described, our system consists of independent parallel machines, which is a very important characteristic. In order to be able to distinguish the processing times on each machine for each order type, we have implemented an array variable in the agent "Job" and initialized it in the *Source* element of each job type (at the moment where they enter the system) as follows:

RED:
entity.processingTime = new int[] {24,32,33,21,21,29,27,32,29};
BLUE:
entity.processingTime = new int[] {34,27,22,26,22,23,29,33,25};
YELLOW:
entity.processingTime = new int[] {28,32,25,26,23,25,21,21,23};

Table 2: Processing times (small) per type on each machine

RED:
entity.processingTime = new int[] {24,32*10,33*6,21,21*6,29*10,27,32*10,29*6};
BLUE:
entity.processingTime = new int[] {34*10,27*6,22,26*6,22*10,23,29*6,33*10,25};
YELLOW:
entity.processingTime= new int[] {28,32*6,25*20,26*10,23,25*6,21,21*6,23*10};

Table 3: Processing times (large) per type on each machine

This means that all machines are denoted from 0 to 8 and depending on which machine a certain job lands on, the corresponding value from the array will be added to the « productive time » and « busy time » of this machine. By collecting these working times and dividing them by the overall makespan we are able to tell the mean utilization of any desired machine either with or without setup consideration. Busy time, in this context, refers to the finite delay time caused by a machine that is equivalent to the sum of processing time (=productive time) and setup time.

In the stochastic version of the model we distribute the processing times with a triangular function. We used in all scenarios a 20% deviation for both the minimum and the maximum values of the triangular distribution: *triangular (entity. processingTime[0]*0.8, entity.processingTime[0], entity.processingTime[0]*1.2)*. This results in the end in a randomly fluctuating processing time for each machine, but with the mean values given in *Table 2&3*. For Scenario 3, we were experimenting with the same distribution function, including 80% deviation but finally applied the values from *Table 3* for the final version of this scenario.

4.1.2. Setup Times

Although we already talked briefly about setup times in Anylogic by means of *Table 1*, the implementation of these times was not quite straightforward, therefore, in order to better understand the logic behind it, we would like to explain this in more detail here. In addition to the two-dimensional array variable "setupTime", we had to define a logic so that the program remembers whether the setup times have to be performed on a particular machine or not. For this reason, we have created another function within the agent Job, which we call "getSetupTime". In this we define that if the colour of the individual entity we are addressing is the same as the colour of the entity just in front of it in the queue of each buffer slot (BS), the setup time is zero. In any other case, whether the job is red, blue or yellow we select the correct value from the "setupTime" matrix and add it to the total delay times of the machines. The values of this particular matrix (*Table 1*) can however be changed any time, as already mentioned. During the experiments of Scenario 4 which is a scenario with highly differentiated setup times, we used significantly elevated values:

	Red	**Blue**	**Yellow**
Red	0	600	500
Blue	600	0	400
Yellow	500	400	0

Table 4: Setup times (large) between different types of jobs

The example of how the "getSetupTime" function is implemented is demonstrated in *Figure 6*:

```
int setupT = 1000;

if (lastColor.equals(this.Type))
    setupT = 0;

else if (this.Type.equals("RED")){
    if (lastColor.equals("BLUE"))
        setupT = setupTime[0][1];
    else
        setupT = setupTime[0][2];
    }

else if (this.Type.equals("BLUE")){
    if (lastColor.equals("RED"))
        setupT = setupTime[1][0];
    else
        setupT = setupTime[1][2];
    }

else if (this.Type.equals("YELLOW")){
    if (lastColor.equals("RED"))
        setupT = setupTime[2][0];
    else
        setupT = setupTime[2][1];
    }
else
    setupT = 666;

return setupT;
```

Figure 6: Calculation of setup times

4.1.3. Transportation Times

(Hybrid) Flow Shop problems with transports have already been discussed in various forms in the literature. However, the focus is often on dealing with practical details of the problem and maximizing the process throughput. Xuan and Tang (2007) present a time-indexed model that takes into account transport times between processing steps. Nevertheless, their problem is greatly simplified, as the capacity of the means of transport is not limited (Gondek, 2011). In contrast, the transport unit(s) are strictly limited to the capacity of transporting a single job in our problem. According to Ribas et al. (2010) both the removal or transportation times are also non-HFS-specific constraints and can be found in any manufacturing environment. It is at least as important as the other two components (processing and setup times) that we have discussed before, because by reducing the transportation times we can significantly reduce the overall makespan.

To implement a transport time of the agent "Crane" from stage to stage and from machine to machine, we have created a global variable, which we have named "transportMatrix". This variable is also a two-dimensional array, that represents the times required by the transport element from point A to point B in the system. However, in order to be able to access the correct elements of the matrix for each job transfer, we had to implement two additional parameters for the "Job" agent, which can then automatically access the desired places in the matrix within the delay time of the service element (=transportMatrix[entity.lastMachine][entity.nextMachine]). These two parameters represent the ‘From’ and the ‘To’ columns and rows of *Table 5* and *6*. These matrices are also symmetrical, similar to the setup matrix. This means that it takes exactly as long to reach a point within the factory as the return from this point to the starting point. The distances given in *Table 5* more or less accurately represent the layout of the designed flow shop. Additionally, we also have a variant (*Table 6*) for our second scenario where the matrix is inspired by the layout as well as before, but where the transport times are extended compared to *Table 5*.

	BS11	BS21	BS31	BS12	BS22	BS32	BS13	BS23	BS33
Q1	1	2	3	4	5	6	9	10	12
Q2	2	1	2	5	6	5	11	13	11
Q3	3	2	1	8	7	6	15	13	10
TS11	4	5	8	3	4	5	3	5	7
TS21	5	6	7	4	2	6	6	5	6
TS31	6	5	6	5	6	3	7	5	4
TS12	9	11	15	3	6	7	2	3	4
TS22	10	13	13	5	5	5	3	2	3
TS32	12	11	10	7	6	4	4	3	1

Table 5: Transport times (short) with respect to the layout of the factory

	BS11	BS21	BS31	BS12	BS22	BS32	BS13	BS23	BS33
Q1	4	6	8	8	10	12	18	20	24
Q2	6	4	6	11	13	11	21	23	21
Q3	8	6	4	16	14	12	25	23	20
TS11	8	11	16	8	10	12	7	10	14
TS21	10	13	14	10	6	16	11	10	11
TS31	12	11	12	12	16	8	14	10	8
TS12	18	21	25	7	11	14	6	8	10
TS22	20	23	23	10	10	10	8	6	8
TS32	24	21	20	14	11	8	10	8	4

Table 6: Transport times (long) with respect to the layout of the factory

For a better understanding: the parameter “lastMachine” keeps track of the machines last visited by the job, so that it counts the machines from 0 to 8 that represent the rows of the matrix. The columns are a bit trickier to display, because for that purpose each order should know beforehand where it is going. To overcome this, it was inevitable to transform the way of the decision making – that the Select Output element was taking care of so far – into a more accessible decision function about where the job should travel next. For this purpose, we have created a global function that returns the number of a particular machine represented by the columns of the matrix, or in other words, where the job is to be transported next. This first global function is called “decideNextMachine” and, to put it simply, decides where the orders

are to be transported. It has two arguments: ‘entity’ of type “Job” in the first place and ‘policy’ of type “String” in the second place. These arguments are passed by another important function that we call “updateNextMachines”, at first for each entity at the top of each queue in the system waiting for the Hold elements to open, and then for the policy rules that we have defined. The latter function is implemented in the already mentioned “update_policyRule” function, in order to be able to fully align all decisions. We will give a more detailed explanation of the decision-making process using *Figure 7* in the next section.

4.2. Outline of the Heuristic Decision-Making Process

Since we had to consider many problems simultaneously, it seemed reasonable to use problem decomposition to approach this study, as so often is the case in heuristic procedures. According to Quadt & Kuhn (2007), heuristic problem-solving methods can be further divided into holistic and decomposition approaches. The first approach, also known as the “integrated approach”, deals with various aspects of a problem simultaneously. Meanwhile, the latter approach breaks down the problem with respect to individual orders (job-oriented decomposition), sub-problems (problem-oriented decomposition: batching, sequencing and loading) and production stages (problem-oriented decomposition). The most common decomposition concept is described by Linn and Zhang (1999) or Wittrock (1988). The first step is to decide which order is to be processed on which machine of a stage. In the second step, the sequence of the orders to be processed is determined for each individual machine. The solution possibilities for the generated partial problems are manifold and depend of course on the objective function under consideration. However, due to the complexity of HFS problems, it is a common practice to segment the problem and treat and successively solve its sub-problems according to the divide and conquer principle (Gondek, 2011). This was however not an option for our simulation, because to create a dynamic environment, we had to solve all sub-problems simultaneously. After Quadt and Kuhn (2007), an example of a simple holistic approach would be to select the next order that must be produced when a machine is idle with the support of scheduling rules. Although we would prefer to describe our simulation as holistic, it was still very important for the implementation of our model to divide the overall problem into smaller problems and to implement these one by one in order to be able to execute them in a meaningful way during simulation. For this reason, we would

like to discuss in the next section how these partial problems have been implemented and how we could effectively handle the interdependencies between the problem segments.

As already mentioned, we had to introduce many decision steps to distribute each job dynamically and synchronously throughout the system. In our example, the two most important decisions that need to be made, regardless of the policy we use, are which order is to be delivered next and where it is to be delivered. Therefore, our schedule must consist of machine assignments for each operation as well as a production sequence for each machine. In addition, the timing of these decisions is very crucial. In the very first draft of the model, we created a number of different decision functions such as one that is responsible for machine assignment, one that is responsible for job sequencing and the policies. Thereafter, we integrated these functions in a two-step procedure, in which we could separately control the prioritization of the respective jobs or even type of jobs, and also the machine policy that should be used. This was a very practical solution for a first draft which had the potential to implement different job priority rules independently from the machine assignments.

Later, however, we found that this two-step procedure in our case makes the structure of the model more difficult than it is simplified. Since we wanted to focus only on time without additional job-related constraints, such as due dates, for which the two-step procedure could have been reasonable, we opted for a more manageable approach. In this current configuration, our job priorities and machine priorities are executed in a single step because they are by all means interdependent. This means that after the machine with the smallest processing/setup/transport/waiting times (PT, ST, TT, WT) has been assigned to an order at the top of each transport slot, we resume the minimum of these times among the waiting orders in the same update function. Finally, the order with the smallest time is transported next. In the scenario with two transport units, we proceed in the same way, with the difference that after each update, two jobs can be forwarded simultaneously, namely the one with the smallest total times and the one with the second smallest. To better illustrate the functions described above, we would like to include a generalized depiction of the implementation of the decision-making process here:

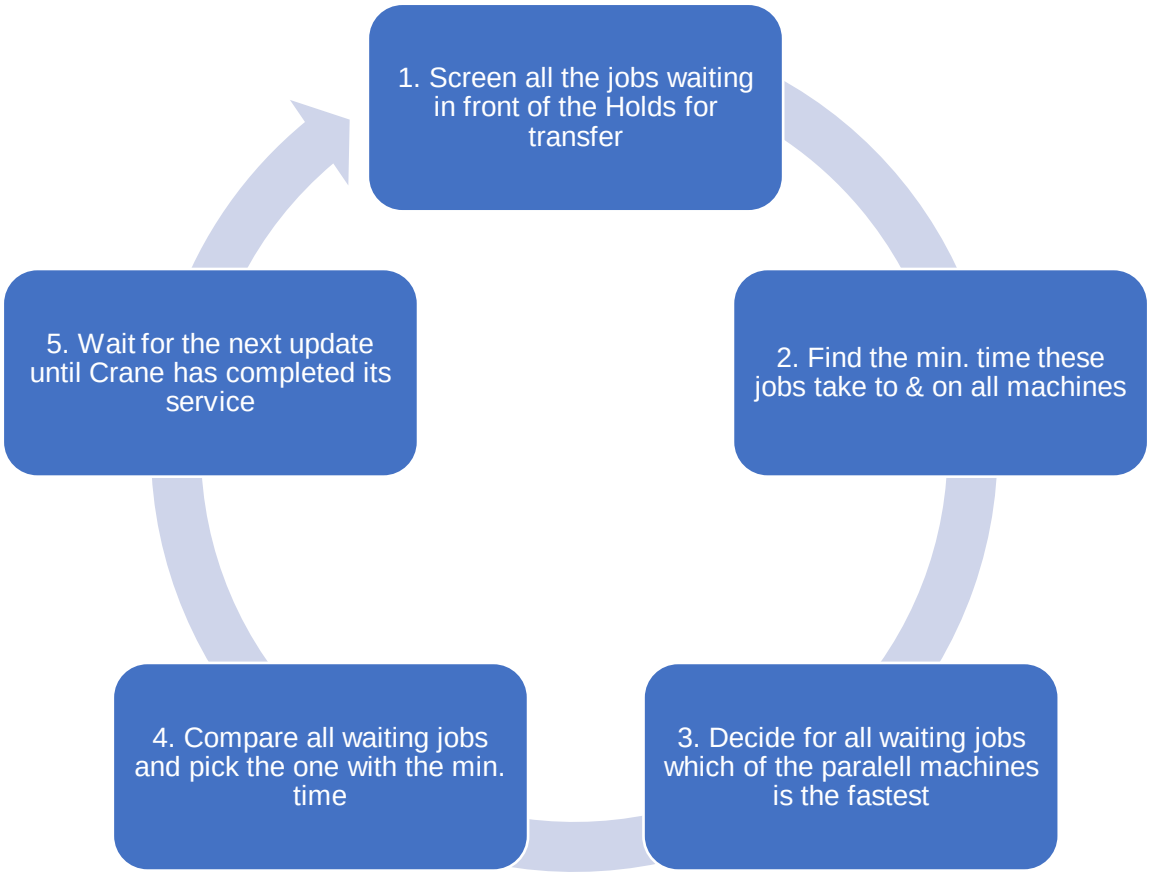

Figure 7: Implementation of the decision-making process

As these decisions must be made in the demonstrated order and must always be synchronised on account of their interdependence, there is another simple function which has been implemented in the "update_policyRule" function: recalculateHolds(). This last step ensures that each time a new job is stopped at the hold element, old values will be forgotten, and the blocking condition will be re-evaluated. Also, we should not neglect that jobs do not forget the conditions that have been assigned to them, unless we make sure that they do. For this reason, their priority is set to false at each time they enter the service element to ensure that no job takes its priority value to the next level.

4.3. Priority & Machine Policies

Before we come to our results, we would like to introduce you to the different dispatching rules that apply directly to all machine assignment and successively to all job sequencing as well. A total of five very basic objectives were implemented (Shortest Queue [0], Shortest Processing Time [1], Shortest Setup Time [2], Shortest Transporting Times [3] and Shortest Waiting Time [4]) and we finally developed 16 job priority- and machine policies from these and their combinations and packed them into an array called “allPolicies”. These policies are explained in more detail here.

0. Shortest Queue

This rule is probably the most self-explanatory of all, therefore will not need to be explained very thoroughly. As its name already implies, this policy focuses on nothing else than on the number of waiting orders in the buffer slots. An arbitrary job which is forwarded with the help of a triangular function (entity.priority_time = (int)triangular (0,5000,10000)) will be chosen by the crane and delivered to the machine with the smallest BS queue at the next stage. On account to this randomness and to the fact that this policy does not include job prioritization, you can recognize in the results displayed that the number of switches between different types of jobs for this rule is always the highest. However, this does not mean that it always performs poorly when it comes to the overall makespan. In fact, in our baseline scenario it even works slightly better than its main competitor (SWT), although there is also a scenario – in which we simulate with much higher setup values (Scenario 4) – in which it performs significantly worse than any other strategy.

1. Shortest Processing Time

From theory, we know that the SPT rule can be optimal to minimize the makespan of a flow shop if we have a proportionate FFS with a single stage or with any number of stages in which there is only one machine at each stage (Pinedo, Flow shops and flexible flow shops (Deterministic), 2016, S. 174-175). However, this only applies to deterministic proportionate flow shops with unlimited storage. In stochastic FFS, we have to talk about shortest *expected* processing times because it is not possible to predict the exact processing times on the machines – as they can vary randomly. It is only possible to talk about the most likely value (an expected value) by using the means of the distributions with which we have chosen to demonstrate the randomness of our system.

For this rule, we have created a function that returns the shortest processing times for each machine stage by stage. We have separated the three processing levels in order to be able to use the simulation also for purposes other than those described in this work, e.g. to see what happens if certain jobs skipped stages. Our first rule (SQ) was not yet able to decide which task to prioritize (it was randomly chosen which entity the transport resource should pick up first). It has only decided to which machine the flow should be continued, depending on the length of the queue in front of it. As opposed to that, this priority rule should not only know where the jobs are to be moved to, but also which jobs are to be transferred.

The main principle of the implementation remains quite similar in case of all policies: we introduce new parameters in the form of an array to the respective function. These parameters are called “processingTimes”, “setupTimes, “transportTimes” and “waitingTimes”. Depending on the function, one or more of these are defined beforehand. These arrays are then filled with “real” time values from our original arrays in a switch-command which of course evaluates their location (using “lastMachine” that displays the machine or the queue they last left) in each update before outputting these values. The times to be taken into account when deciding which path a particular job should take are defined by a variable called “timesNeeded”, which corresponds to the times we look at stage by stage. In a last step, the target machine will be selected with the command “indexOfMin(timesNeeded)”, while the smallest times that give the jobs their priority values are stored in a job parameter with the command “timesNeeded[targetMachine]”.

2. Shortest Setup Time

Although these rules try to achieve different goals (apart from reducing the expected completion time) – like reducing the overall setup times in this policy case or shortening the transport distances in the STT rule – and therefore are very different, the logic of the implementation is the same when we observe the code. The interesting thing about this rule is the way we managed to access the right setup times from our setup matrix, because it was not sufficient to simply take the indices of the machines into account, as we did with SPT. In this case we had to know two things: which type of job was last on the machine and which is next. For this, we considered the type of job that last left the machines for which the next job is waiting. The "lastJobColors" array symbolizes the types of jobs and serves to compare the upcoming job with itself. If they coincidentally match, no setup time is added to the "timesNeeded" calculation. If any other colour appears, the value of the respective index of the matrix will be returned by the function "getSetupTime" and entered as a value that helps to find the target machine in this rule.

3. Shortest Transport Time

In order to access the time span that orders require for a transport from A to B (= "timesNeeded"), we needed a simple solution to find the correct indices of our already mentioned transport matrix at each point in time during the simulation. As we set up our model, the transport times that we considered at the beginning consisted only of the times that a certain order has to spend on the transport unit in order to be transported from one machine to another. This means that one half of the matrix was not considered at all and that the distance our crane had to travel back and forth between stages before it could pick the next order was never accounted for. We can also imagine this as if the crane always stood in front of every machine, ready to deliver to the next stage, but never needed time to reach or return from a machine on another stage, just as the units of a resource pool in AnyLogic are seized and released in "no time". This was easy to implement, but of course very unrealistic. After some considerations, we were able to add up the duration of two transport phases for all scenarios: when the crane transports an order and also when it travels empty. In all current scenarios (1-5), the transport time therefore corresponds to the arrival time (of the crane) plus delivery time.

To calculate the first phase, we located our entities using the information obtained by the variable “lastMachine” as starting points and the machines as arrival points. For the latter, we had to take the last position of the transport unit, where it dropped the last order it transported as a starting point and the location defined by the variable “lastMachine” as an arrival point. Once we have received these aggregated delay times, we can proceed as in any other strategy described: identifying the target machine and memorizing the smallest delay time as a priority value for all orders to be considered later.

```
int[] transportTimes = new int[3];

int[] timesNeeded = new int[3];

int location = entity.lastMachine;


// get the transport times for all possible next machines
//... + transportMatrix[entity.lastMachine][lastPosition_Crane]

switch(location){
case 0: ;
case 1: ;
case 2: transportTimes[0] = transportMatrix[location][0] + transportMatrix[lastPosition_Crane][location];
        transportTimes[1] = transportMatrix[location][1] + transportMatrix[lastPosition_Crane][location];
        transportTimes[2] = transportMatrix[location][2] + transportMatrix[lastPosition_Crane][location];
        break;

case 3: ;
case 4: ;
case 5: transportTimes[0] = transportMatrix[location][3] + transportMatrix[lastPosition_Crane][location];
        transportTimes[1] = transportMatrix[location][4] + transportMatrix[lastPosition_Crane][location];
        transportTimes[2] = transportMatrix[location][5] + transportMatrix[lastPosition_Crane][location];
        break;

case 6: ;
case 7: ;
case 8: transportTimes[0] = transportMatrix[location][6] + transportMatrix[lastPosition_Crane][location];
        transportTimes[1] = transportMatrix[location][7] + transportMatrix[lastPosition_Crane][location];
        transportTimes[2] = transportMatrix[location][8] + transportMatrix[lastPosition_Crane][location];
        break;

default:
        transportTimes = null;
        System.out.print("\n****ERROR: WRONG LOCATION****\n");
        break;
}

// ADD ALL TO FIND OUT THE OVERALL TIME THAT IS NEEDED ON EACH POSSIBLE MACHINE

timesNeeded[0] = transportTimes[0];
timesNeeded[1] = transportTimes[1];
timesNeeded[2] = transportTimes[2];
```

Figure 8: Calculation of transport times

4. Shortest Waiting Time

Before concluding the description of the rules with one prominent example, where all policies are combined in our smartest and supposedly most powerful policy (rule 16), we would like to devote a few words to the last basic rule we have created for this problem in order to positively influence the outcome. This policy was supposedly the hardest to implement properly. But before we go into details: what was the idea behind and what did we want to accomplish? The SWT policy should predict where each job should travel next immediately prior to its decision, not only by the length of queues in front of the machines (as in the SQ policy), but also by the minimum waiting time they have by entering a particular queue. The idea was to consider here the number of the elements in the queue times the processing time and setup time they will take in the machine respectively.

For this rule, we have tried several workarounds, which seemed to perform quite well, e.g. the number of elements in the queues times the overall mean processing times and mean setup times. However, we realised that this way of calculation would not deliver us very accurate results in Scenario 3, where we decide to differentiate the machine processing times between machines by 10 or even 20 times. Especially the unpredictability of random processing times is also a contributing factor. Finally, we found a way to calculate the exact processing times and setup times of every order in every queue by two separate functions:

```
int overallPT = 0;
int overallST = 0;


for( int i=0; i < q.size(); i++){
    overallPT = overallPT + q.get(i).processingTime[machineIndex];


    if (i==0)
        overallST = overallST + q.get(i).getSetupTime(ColorOfMachine[machineIndex]);
    else
        overallST = overallST + q.get(i).getSetupTime(q.get(i-1).Type);


}


return overallPT + overallST;
```

Figure 9: Calculation of waiting times I – getQueueWaitingTime()

```
int[] waitingTimes = new int[3];


if(stage == 1){
    waitingTimes[0] = getQueueWaitingTime(BS11, 0);
    waitingTimes[1] = getQueueWaitingTime(BS21, 1);
    waitingTimes[2] = getQueueWaitingTime(BS31, 2);

}

else if(stage == 2){
    waitingTimes[0] = getQueueWaitingTime(BS12, 3);
    waitingTimes[1] = getQueueWaitingTime(BS22, 4);
    waitingTimes[2] = getQueueWaitingTime(BS32, 5);

}

else{
    waitingTimes[0] = getQueueWaitingTime(BS13, 6);
    waitingTimes[1] = getQueueWaitingTime(BS23, 7);
    waitingTimes[2] = getQueueWaitingTime(BS33, 8);

}

return waitingTimes;
```

Figure 10: Calculation of waiting times II – getWaitingTimes()

As it is illustrated in *Figure 9*, processing times are calculated using the PT array while to get setup times, the first element of the queue (which enters Processing next) compares its type with the type of order on the machine, whereas every other order compares its type with the type of order right in front of it. Next, in *Figure 10*, we can see how the above function is converted into waiting times by handing over its two arguments, queue (BS-buffer slot) and machine index (0-8). If there is absolutely no job in the queue, waiting time stays zero, and the job with the minimum waiting time in the whole system gets to be transported next. It is also worth mentioning here why we only and exclusively opt for the waiting times created by the elements of the respective queues, and not also for those of the jobs, from which perspective we decide. This is because if we combined this policy with another that included either SPT or SST, we would "charge" twice the time to process and set up the job, once in SPT/SST and once in SWT.

5. Combined policies

All policies so far strictly separated parts of the problem. Thanks to this differentiation, they could be combined into more complex rules without taking aspects into account more than once. Another important requirement for the combination of rules was also the fact that, with the exception of SQ, we used the same unit (=time) for decision-making in each rule. After we had recognized this, we came up with the idea to combine these rules in further functions and to use the values they have returned in aggregated form for the creation of new machine policies. This is the reason why we will talk about some rules as "dominators" of others in certain combinations depending on the scenario. A very good example of this could be the STT+SWT rule. If we have really long transport times, this means that their value will outweigh the value of significantly smaller processing and setup times contained in WT. This means that it is much less important to deliver the jobs to the machines where the actual waiting times are short then to deliver them to the machines which are closest. As you will see in Scenario 2 in section 5.3, colour switches in STT+SWT are reduced to approximately a quarter of the switches of the basis scenario due to longer transport times between the types of orders since they are not initially pooled at one location but picked up from three different locations (Q1, Q2, Q3). Observing the described example during simulation, we can see how the crane's preferences change over very long transport times. Instead of delivering all kinds of jobs to different machines, mainly red and blue jobs are delivered first and the yellow ones at the end, since it would simply take too long to pick up a yellow order from the upper three machines when the crane is stationary there.

In order to demonstrate how these combinations work in practice, we would like to outline here in *Figure 9* the functioning of our most complex combination that occupies the last index of our "allPolicies" array, the 16th rule: SPT+SST+STT+SWT, which we might also refer to as the "rainbow" rule, as it takes into account all the parameters we have just described.

```
// ADD ALL TO FIND OUT THE OVERALL TIME THAT IS NEEDED ON EACH POSSIBLE MACHINE

timesNeeded[0] = processingTimes[0] + transportTimes[0] + setupTimes[0] + waitingTimes[0];
timesNeeded[1] = processingTimes[1] + transportTimes[1] + setupTimes[1] + waitingTimes[1];
timesNeeded[2] = processingTimes[2] + transportTimes[2] + setupTimes[2] + waitingTimes[2];

// Find minimum value and target machine

int targetMachine = indexOfMin(timesNeeded);
int smallestTime = timesNeeded[targetMachine];


switch(location){

case 0:
case 1:
case 2:
        break;

case 3:
case 4:
case 5: targetMachine = targetMachine + 3;
        break;

case 6:
case 7:
case 8: targetMachine = targetMachine + 6;
        break;
}


entity.priority_time = smallestTime;

return targetMachine;
```

Figure 11: Finding the target machine in policy SPT+SST+STT+SWT

5. Results

5.1. Parameters Variation – The experimental setup

Before presenting the results of the different scenarios, I would like to recapitulate the most important things about the experiments in this part:

A parameters variation is an experiment provided by AnyLogic that executes the complex model simulation in the form of several individual model runs that vary one or more "root object parameters". In our parameter variation experiment, we set ten replications for each parameter value, with each replication consisting of simulating the operation of the system to complete 3000 jobs. With this experiment we were able to compare the behaviour of the model with different parameter values (in our case: "policyIndex") and analyse how the different rules affect the model behaviour. Additionally, the use of such an experiment with fixed parameter values within one run (= 10 repetitions in our case) also facilitates the estimation of the influence of stochastic processes (AnyLogic, 2018).

The results for the analysis of our performance indicators will be summarized in a standardized table for each scenario in the next section, which contains the *average total completion time* (of each job type and in total), the *machine utilization* (of each machine and in total) and finally the *number of setups* (for each machine and in total). The mean of the expected total completion time $E(C_{max})$ is computed by AnyLogic's Statistics objects, which calculate statistical information (mean, minimum, maximum, etc.) on a series of data samples of type double for each machine and for the entire system. Information will be added about the time spent in the system each time a job enters the sink object. After each iteration, we search for the information about the maximum time the different order types spent in the system and summarize it in the tables represented by the average of the 10 replications. In addition, for better perception, we have converted the model output given in seconds into hours for this indicator. The other two criteria also work very similarly; however, it is very important to mention that for the machine utilization criteria, we only record the time that a machine has processed an order, without counting setup times, in order to be able to examine and maximize the "sensibly" used utilization times of the machines. When a machine is "utilized" in our understanding here, it automatically signifies that it is also productive in that time, because we do not count the time of setups to the utilization time (= referred to as

productivity from now on). However, by means of another variable ("busy times") and our time plots we were able to keep track of the workload of machines over time including setups for certain rules that interested us. These plots are also displayed in hours. The overview tables will help us to compare individual details of the scenarios and to visualise the differences between all policies.

It should also be emphasized that all scenarios were simulated on the basis of Scenario 1. In this scenario the input data were partly taken from real-world data (see setup times + processing times) provided by the consultancy for the aforementioned project of our faculty and partly inspired and complementary created by them (see transport times). The composition of Scenario 1 is only changed by individual parameters in all other scenarios. This is also the reason why in parallel with the analysis of the performance of individual rules within the scenarios, we will use this scenario to measure the influence of individual parameters on the system as a point of comparison.

5.2. Scenario 1 – Scheduling with basic, real-world inspired parameters

In the first instance, we would like to discuss the so-called "baseline" scenario. This scenario was set up with the following key parameters: setup times (small) according to *Table 1*, processing times (small) according to *Table 2* and finally the transport times (small) according to *Table 5*. When we take a look at the results shown in *Table 7*, we can see the best rules marked green for our three most important evaluation criteria: makespan, capacity utilization and the number of setups. What becomes clear at the first glance when looking at the outcome is that – at least in this scenario – there is an indisputable meaning to every rule that supports the distribution of jobs between the machines. All policies that are marked red due to their long total completion time, are policies without the shortest waiting time or the shortest queue consideration. These policies (SQ and SWT) were designed to have some kind of distribution function by sending jobs to all kinds of machines (with no or little waiting time) instead of focusing on sending them only to those that have the fastest processing and/or setup times or to those that are the closest. As illustrated, the fastest policy (STT+SWT) manages to dispatch all 3000 jobs within 13.8 hours which counts almost twice as fast as the slowest rule (SPT+SST) with only 25.1 hours. Besides, it is also apparent that the only policy which incorporates SWT, but nonetheless turns out to be very slow is the policy where we

combine SWT with the least favourable rule (SPT+SST+SWT). This leads us to conclude that these parameters are rather negligible in the basic scenario, and transport times have much more meaning, even when they are kept small. Our second-best rule, SST+STT+SWT, includes an additional parameter in the dispatching decision: setup times. This second place is not so surprising when we look at the productivity of this policy, which competes with the winning one. When we consider the results shown by the first four rules that only incorporate individual parameters (PT/ST/TT/WT), we can determine that directly after SWT and then STT, SST works best.

	SQ	SPT	SST	STT	SWT	SPT+SST	SPT+STT	SST+STT	SPT+SWT	SST+SWT	STT+SWT	SPT+STT+SWT	SPT+SST+SWT	SST+STT+SWT	SPT+SST+STT	SPT+SST+STT+SWT
Red_makeSpan	20,3	16,7	7,5	10,0	6,5	10,9	11,3	9,2	19,0	6,1	7,4	11,7	5,9	4,9	10,0	9,9
Blue_makeSpan	20,2	15,9	23,9	16,0	12,6	16,0	13,8	16,0	9,1	20,5	10,6	8,6	14,8	14,5	15,9	12,3
Yellow_makeSpan	20,3	25,1	14,6	22,8	20,4	25,1	20,6	23,2	23,0	14,0	13,8	15,9	23,6	13,1	18,6	14,4
MakeSpan	**20,4**	**25,1**	**23,9**	**22,8**	**20,4**	**25,1**	**20,6**	**23,2**	**23,0**	**20,5**	**13,8**	**15,9**	**23,6**	**14,5**	**20,6**	**15,9**
M0 Utilization	0,7	0,3	1,0	0,3	0,8	0,3	0,3	0,3	0,3	0,8	0,6	0,6	0,3	0,6	0,3	0,5
M1 Utilization	0,4	0,0	0,0	0,3	0,3	0,0	0,0	0,4	0,0	0,3	0,6	0,1	0,1	0,6	0,0	0,1
M2 Utilization	0,1	0,5	0,0	0,3	0,1	0,5	0,6	0,2	0,5	0,1	0,4	0,6	0,5	0,4	0,6	0,6
M3 Utilization	0,7	0,2	0,8	0,3	0,8	0,2	0,3	0,3	0,3	0,8	0,5	0,4	0,2	0,5	0,3	0,4
M4 Utilization	0,3	0,5	0,0	0,3	0,2	0,5	0,0	0,3	0,5	0,2	0,5	0,2	0,5	0,5	0,0	0,2
M5 Utilization	0,0	0,0	0,0	0,3	0,0	0,0	0,6	0,2	0,0	0,0	0,4	0,6	0,0	0,4	0,6	0,6
M6 Utilization	0,7	0,5	0,9	0,3	0,8	0,5	0,4	0,3	0,4	0,8	0,6	0,5	0,5	0,5	0,4	0,5
M7 Utilization	0,3	0,0	0,0	0,4	0,2	0,0	0,3	0,6	0,1	0,2	0,6	0,3	0,0	0,7	0,0	0,2
M8 Utilization	0,0	0,3	0,0	0,3	0,0	0,3	0,3	0,0	0,4	0,0	0,4	0,5	0,4	0,4	0,6	0,6
Utilization	**0,4**	**0,3**	**0,3**	**0,3**	**0,4**	**0,3**	**0,3**	**0,3**	**0,3**	**0,4**	**0,5**	**0,4**	**0,3**	**0,5**	**0,3**	**0,4**
M0 Setups	1190,5	0,0	2,0	0,0	2,0	0,0	0,0	0,0	1,0	2,0	2,0	1,0	1,0	2,2	0,0	1,4
M1 Setups	615,9	0,0	0,0	1,0	2,0	0,0	0,0	1,2	3,0	2,0	274,0	13,9	1,1	1,8	0,0	1,6
M2 Setups	165,3	2,0	0,0	1,0	1,3	2,0	2,0	0,8	2,0	2,0	321,8	2,0	2,0	2,0	4,3	4,4
M3 Setups	1344,5	0,0	2,0	0,0	4,6	0,0	0,0	0,0	0,0	2,0	2,8	0,6	0,0	4,3	0,0	0,5
M4 Setups	595,3	2,0	0,0	1,0	3,0	2,0	0,0	1,2	11,6	2,0	276,6	27,6	2,6	6,8	0,0	6,8
M5 Setups	33,3	0,0	0,0	1,0	1,0	0,0	2,0	0,8	2,0	1,2	322,6	2,0	2,0	4,8	4,3	4,6
M6 Setups	1375,0	1,4	2,0	0,0	5,4	1,0	0,0	0,0	511,0	2,0	22,6	169,6	2,2	20,1	0,0	8,3
M7 Setups	564,7	0,0	0,0	1,0	3,8	0,0	1,0	5,4	1,0	2,0	285,2	15,5	0,5	7,1	0,0	2,1
M8 Setups	36,3	1,0	0,0	1,0	1,0	1,0	1,0	0,0	3,4	1,0	336,5	75,6	1,0	5,0	4,3	8,0
No. Setups	**657,9**	**0,7**	**0,7**	**0,7**	**2,7**	**0,7**	**0,7**	**1,0**	**59,4**	**1,8**	**204,9**	**34,2**	**1,4**	**6,0**	**1,4**	**4,2**

Table 7: Results of Scenario 1

The overall utilization of the machines and the number of setups are also worth mentioning, although, as we have already pointed out, the two should not be interlinked. As we will see from every outcome, the workloads in our context always reflect the completion times, which

means that the rules with the highest productivity will always* result in the shortest completion times. Now, when we return to our winning rule and evaluate utilization, we see that it is the rule that has the highest productivity, but also the second highest average number of setups of all rules. We can pronounce this rule the best because of its high productivity, but we should also remember that this high number of settings probably consumes additional machine capacity, which is not shown here in *Table 7*, but could ultimately be interesting to see.

Therefore, we have prepared two time plots for the utilization of rules 10 (STT+SWT) and 13 (SST+STT+SWT), in which we have included the setup times in our calculations for the utilization. Both time plots show us the machine load including setups for all 9 machines with different colours over time. As we can see in *Figure 12*, the utilization of the different machines, although very different, remains more or less constant over time. The utilization starts to decline after ca. 10 hours (except M5 & M8) and about 13.8 the last jobs are leaving the system. On the other side of *Figure 13* we can observe an obviously different course. In this process, at the beginning only certain machines are used to full capacity, while in the second phase the load on all machines is approximately equalized. This means that machines that have been used little are used more and those that were used much are used less. This may be because the red orders are processed first (this is the default colour of all machines) on the nearest machines, while the other two types are shipped in parallel and distributed approximately equally on the machines. However, in this policy the emphasis is on the good distribution of jobs without them being shuffled all the time, thanks to the fact that with SST we pay special attention to setup times.

Since we have many jobs and long working hours (13-25 hours) and our set-up times in this scenario are very short (8-12 seconds), the differences are not so extreme between utilizations calculated with and without setup times. However, an "unproductive" use of the machines due to the additional setups is not always neglectable, especially if we were able to reduce the number of setups from approx. 204.9 to approx. 1.4 against only an additional 0.7 hours due to production with additional SST analysis. With this effort (including SST), we finally found that the productivity of the machine would be reduced by just about nothing (less than 1%) between the best and second-best rules, but the unwanted busy time could be reduced by

* see more: Scenario 3

almost 10%. The decision, however, which rule we define as the best completely depends on the criteria that are important to us, to deliver the orders earlier and on time or, for example, to consider and reduce disproportional machine loads.

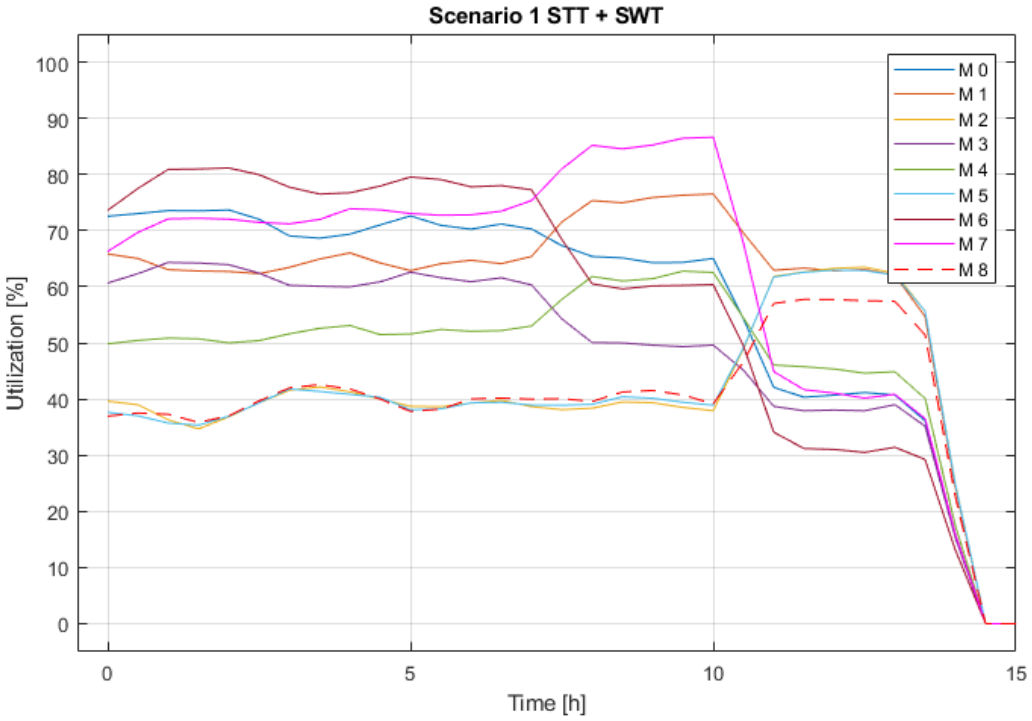

Figure 12: Utilization over time including setups I

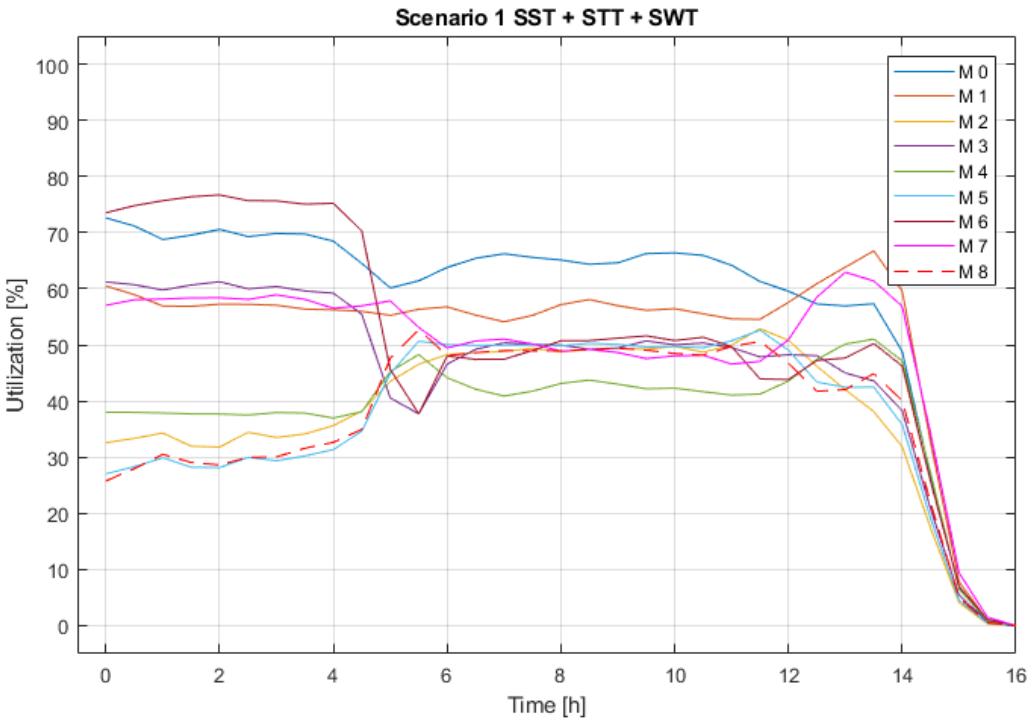

Figure 13: Utilization over time including setups II

Finally, it should also be noted that even for the best and second-best policies (STT+SWT and SST+STT+SWT) in this scenario, the machines are productive on average only half of the respective production times, which led us to believe that the transport is "too slow" in comparison with the other parameters (PT and ST) and that productivity could be higher with a faster transport unit. This assumption can be confirmed by other scenarios to which we will come back later.

5.3. Scenario 2 – Scheduling with Large TT

For our second experiment we left everything unchanged except the transport matrix. Transport times have increased significantly, as shown in *Table 6*. By this change we hoped to confirm the assumption of the importance of the transport times of the construction in the baseline scenario. And indeed, the results indicate without doubt that with long transport times, these alone should now be taken into account (STT). On the first place, as displayed, STT has arrived this time, however, clearly every policy which includes STT marked green is among the fastest rules. This scenario suggests that the transport unit is our bottleneck, however the differences in the expected completion times are not that high between the best (STT) and worst (SPT+SWT) rules as before. Also, SWT seems to lose its importance somehow, and e.g. to complete all jobs using policy STT+SWT takes longer, then any other 2-combination with the rule STT, such as SPT+SWT and SST+SWT. Even among all rules where not two but three parameters are considered, all combinations with SWT turn out to be slower than the one without, SPT+SST+STT. This could be explained by the fact that SWT – which tends to distribute jobs to all types of machines – works against the natural behaviour of STT, which wants jobs to travel only to the nearest machines.

As the completion time increase, the productivity of the machines sinks, considering that not only within scenarios, but also in between them. In this case, the maximum average utilization rate lies around only 20%. As briefly mentioned in an example, the number of setups in policy STT+SWT, which triumphed in our baseline scenario decreased to only ~44 instead of ~205 due to the fact that STT became so dominant within this combination by raising the transport matrix values. On the other hand, however, what was quite unexpected was the increase in the number of type changes in the SPT+STT rule, which rose from an average of 0.7 to 157.7. This could be explained by a conflict between blue and yellow orders which are distributed alternately after the red orders are finished, while in the basic scenario the entire blue batch was processed before the yellow could have started. Since the transports were not yet so dominant, blue jobs have been routed through the fastest machines (M2&M5&M8). In the current scenario, thanks to the new long transport routes, they are sent alternately with yellow to the fastest *and* to the nearest machine in turns.

	SQ	SPT	SST	STT	SWT	SPT+SST	SPT+STT	SST+STT	SPT+SWT	SST+SWT	STT+SWT	SPT+STT+SWT	SPT+SST+SWT	SST+STT+SWT	SPT+SST+STT	SPT+SST+STT+SWT
Red_makeSpan	44,2	33,2	13,1	10,7	13,5	13,1	11,2	10,7	33,3	13,1	14,5	16,6	13,1	11,8	11,2	12,1
Blue_makeSpan	43,8	33,2	43,9	22,2	27,8	33,2	34,5	22,2	27,0	44,1	23,4	33,2	33,2	30,0	31,8	29,0
Yellow_makeSpan	44,0	53,7	29,6	33,2	44,1	53,7	27,2	34,3	53,7	29,6	34,8	36,3	53,1	34,0	24,8	35,4
MakeSpan	**44,3**	**53,7**	**43,9**	**33,2**	**44,1**	**53,7**	**34,5**	**34,3**	**53,7**	**44,1**	**34,8**	**36,3**	**53,1**	**35,5**	**34,0**	**38,3**
M0 Utilization	0,5	0,1	0,5	0,2	0,5	0,1	0,2	0,2	0,1	0,5	0,3	0,2	0,2	0,3	0,2	0,2
M1 Utilization	0,1	0,0	0,0	0,2	0,0	0,0	0,0	0,5	0,0	0,0	0,3	0,1	0,0	0,3	0,0	0,1
M2 Utilization	0,0	0,2	0,0	0,2	0,0	0,2	0,4	0,0	0,2	0,0	0,1	0,2	0,2	0,1	0,4	0,3
M3 Utilization	0,4	0,1	0,5	0,2	0,5	0,1	0,2	0,2	0,1	0,5	0,2	0,2	0,1	0,2	0,2	0,1
M4 Utilization	0,0	0,2	0,0	0,2	0,0	0,2	0,0	0,4	0,2	0,0	0,2	0,1	0,2	0,2	0,0	0,2
M5 Utilization	0,0	0,0	0,0	0,2	0,0	0,0	0,4	0,0	0,0	0,0	0,1	0,2	0,0	0,1	0,4	0,2
M6 Utilization	0,5	0,2	0,5	0,2	0,5	0,2	0,2	0,2	0,2	0,5	0,3	0,2	0,2	0,2	0,2	0,3
M7 Utilization	0,0	0,0	0,0	0,3	0,0	0,0	0,0	0,4	0,0	0,0	0,3	0,0	0,0	0,3	0,0	0,0
M8 Utilization	0,0	0,1	0,0	0,2	0,0	0,1	0,4	0,0	0,1	0,0	0,1	0,3	0,1	0,1	0,4	0,2
Utilization	**0,2**	**0,1**	**0,2**	**0,2**	**0,2**	**0,1**	**0,2**	**0,2**	**0,1**	**0,2**	**0,2**	**0,2**	**0,1**	**0,2**	**0,2**	**0,2**
M0 Setups	1735,9	0,0	2,0	0,0	2,0	0,0	0,0	0,0	0,0	2,0	2,0	1,2	1,0	1,8	0,0	1,0
M1 Setups	219,4	0,0	0,0	1,0	1,0	0,0	0,0	2,0	0,0	2,0	114,6	1,2	0,0	2,8	0,0	1,0
M2 Setups	10,2	2,0	0,0	1,0	0,0	2,0	473,2	0,0	2,0	0,2	14,6	694,2	1,9	1,3	6,4	3,4
M3 Setups	1887,7	0,0	2,0	0,0	4,0	0,0	0,0	0,0	0,0	2,0	2,0	1,2	0,0	1,0	0,0	0,0
M4 Setups	89,3	2,0	0,0	1,0	0,3	2,0	0,0	2,0	2,0	0,6	114,6	194,8	2,2	150,8	0,0	3,3
M5 Setups	0,0	0,0	0,0	1,0	0,0	0,0	473,2	0,0	0,0	0,0	14,6	694,0	0,0	1,0	6,4	2,6
M6 Setups	1923,7	1,0	2,0	0,0	4,0	1,0	0,0	0,0	4,8	2,0	2,6	51,3	1,0	0,8	0,0	1,3
M7 Setups	64,3	0,0	0,0	1,0	0,0	0,0	0,0	2,0	0,2	0,3	117,6	1,0	0,0	1,5	0,0	1,1
M8 Setups	0,0	1,0	0,0	1,0	0,0	1,0	473,2	0,0	1,1	0,0	14,6	947,2	1,0	10,9	6,4	4,8
No. Setups	**658,9**	**0,7**	**0,7**	**0,7**	**1,3**	**0,7**	**157,7**	**0,7**	**1,1**	**1,0**	**44,1**	**287,3**	**0,8**	**19,1**	**2,1**	**2,1**

Table 8: Results of Scenario 2

5.4. Scenario 3 – Scheduling with Large PT

As far as processing times are concerned, we had two options: on the one hand, the observation of the model's reaction to higher fluctuations in processing times e.g. instead of 20% with 80% variance to the mean, while on the other hand, there was the possibility to introduce completely new means to the triangular distributions of the PT. After testing and observing both, we decided to introduce to you the latter possibility, since the important results remained unchanged and the standard deviation also seemed very insignificant due to the high number of jobs having their means unchanged. The idea behind the changed means of the PT between the machines was a threshold investigation and was based on a constellation of manufacturing processes in which the machines of one stage represent very substantial differences for different types of orders. The reason for this is that, even if they are accessible to every colour, each of them is "specialized" for a certain one. This can be seen by comparing the PT of the colours on one machine (see *Tables 2* and *3*). For example, although M2 (on the third index of the array of machines) can process all types, it obviously has a special knowledge required for type blue, as this type can be processed by it much faster than any other one.

In terms of results, we identified the biggest absolute differences between the individual rules within this scenario. Our slowest rule this time is more than 10 times slower than the winning one. This means that in this particular case it is extremely important that the jobs land on the right machines, which is not the case with all rules that contain STT or SST, but not SPT. All combinations without SPT are marked in red with the exception of SWT. Not surprisingly, marked red is also our Shortest Queue policy, since the machine assignment happens in here somewhat randomly. On the other hand, as already mentioned, SWT seems to perform fairly well, probably because SPT is implicitly taken into account in this policy, as processing times are included in waiting times. In addition, it seems that considering how long it takes for elements of a queue to be processed is even better than considering the time a single job takes on a particular machine on its own.

The fact that our “most powerful" policy wins the competition in this setting leaves room for several different interpretations. Nonetheless, among the many interpretations, there is one that we can no longer name in answer to the question of why our rainbow rule (which reflects all possible parameters) is the fastest of all: the high productivity of the machines.

	SQ	SPT	SST	STT	SWT	SPT+SST	SPT+STT	SST+STT	SPT+SWT	SST+SWT	STT+SWT	SPT+STT+SWT	SPT+SST+SWT	SST+STT+SWT	SPT+SST+STT	SPT+SST+STT+SWT
Red_makeSpan	55,2	15,0	7,5	7,5	6,3	8,2	12,5	7,5	14,2	6,6	8,7	11,1	14,2	6,3	12,5	11,1
Blue_makeSpan	55,0	14,2	171,5	98,6	13,3	14,2	13,8	98,6	7,5	21,0	32,7	9,7	7,5	31,9	13,8	9,8
Yellow_makeSpan	55,2	22,4	79,1	234,4	20,9	22,4	21,7	226,0	22,2	14,4	37,8	19,7	22,2	37,1	21,4	19,6
MakeSpan	**55,4**	**22,4**	**171,5**	**234,4**	**20,9**	**22,4**	**21,7**	**226,0**	**22,2**	**21,0**	**37,8**	**19,7**	**22,2**	**37,1**	**21,4**	**19,6**
M0 Utilization	0,9	0,6	0,6	0,0	0,9	0,6	0,7	0,0	0,6	0,9	0,9	0,7	0,6	1,0	0,7	0,7
M1 Utilization	0,9	0,0	0,0	0,2	0,9	0,0	0,0	0,2	0,0	0,9	1,0	0,0	0,0	1,0	0,0	0,0
M2 Utilization	0,9	0,3	0,0	0,6	0,9	0,3	0,3	0,5	0,3	0,9	1,0	0,3	0,3	1,0	0,3	0,3
M3 Utilization	0,8	0,3	0,7	0,0	0,9	0,3	0,3	0,0	0,3	0,9	0,8	0,3	0,3	0,8	0,3	0,3
M4 Utilization	0,7	0,3	0,0	0,3	0,8	0,3	0,3	0,3	0,3	0,8	0,9	0,3	0,3	0,9	0,3	0,3
M5 Utilization	0,6	0,3	0,0	0,2	0,4	0,3	0,3	0,2	0,3	0,4	0,5	0,3	0,3	0,5	0,3	0,3
M6 Utilization	0,6	0,6	0,4	0,0	0,9	0,6	0,6	0,0	0,6	0,9	0,9	0,7	0,6	0,8	0,6	0,7
M7 Utilization	0,9	0,0	0,0	0,4	0,7	0,0	0,0	0,6	0,0	0,7	1,0	0,0	0,0	1,0	0,0	0,0
M8 Utilization	0,5	0,3	0,0	0,3	0,6	0,3	0,3	0,0	0,5	0,6	0,7	0,4	0,5	0,6	0,3	0,4
Utilization	**0,8**	**0,3**	**0,2**	**0,2**	**0,8**	**0,3**	**0,3**	**0,2**	**0,3**	**0,8**	**0,8**	**0,3**	**0,3**	**0,8**	**0,3**	**0,3**
M0 Setups	902,0	1,0	2,0	0,0	2,0	1,0	1,0	0,0	1,0	2,0	2,0	1,0	1,0	2,0	1,0	1,0
M1 Setups	568,7	0,0	0,0	1,0	2,0	0,0	0,0	1,1	0,0	2,0	84,8	0,0	0,0	2,1	0,0	0,0
M2 Setups	519,7	1,0	0,0	1,0	2,0	1,0	1,0	0,9	1,0	2,0	370,4	1,0	1,0	1,7	1,0	1,0
M3 Setups	757,7	0,0	2,0	0,0	3,4	0,0	0,0	0,0	0,0	2,6	62,8	0,0	0,0	3,2	0,0	0,0
M4 Setups	743,1	1,0	0,0	1,0	2,4	1,0	1,0	1,1	1,0	4,8	69,4	1,0	1,0	2,0	1,0	1,0
M5 Setups	480,0	1,0	0,0	1,0	3,4	1,0	1,0	0,9	1,0	3,4	347,5	1,0	1,0	2,3	1,0	1,0
M6 Setups	1114,8	1,0	2,0	0,0	5,8	1,0	1,0	0,0	3,0	4,4	139,0	3,0	2,6	4,8	1,0	2,8
M7 Setups	443,2	0,0	0,0	1,0	2,8	0,0	0,0	3,2	0,8	2,0	22,1	0,6	0,5	2,3	0,0	0,3
M8 Setups	420,6	1,0	0,0	1,0	4,0	1,0	1,0	0,0	2,0	2,8	340,9	2,0	2,0	84,7	1,0	2,0
No. Setups	**661,1**	**0,7**	**0,7**	**0,7**	**3,1**	**0,7**	**0,7**	**0,8**	**1,1**	**2,9**	**159,9**	**1,1**	**1,0**	**11,7**	**0,7**	**1,0**

Table 9: Results of Scenario 3

Therefore, we would like to return to an earlier statement from Scenario 1, namely that the rule with the highest productivity should “always” result in the shortest time span. As you can see in *Table 9*, however, this scenario seems to be the exception to the rule, as this hypothesis does not hold. Namely, while our best rule can work with an average productivity of 30% as the best rule, we have the STT+SWT rule again, which is still very productive, but even at 80% it finishes pretty close to the end of the ranking. How is this possible? To answer this question, we again called up the time plots as a tool, although this time we were only interested in the productivity of the machines. If we now take a look at *Figures 13* and *14*, we see a remarkable difference. This difference lies in the distribution of orders on the machines. As I mentioned earlier, this is very important because we see at first sight what happens if the machine allocation is not successful: We will conclude with about 200 working hours for the 3000 jobs. This is not the case with STT+SWT or SQ, but the reason for their "failure" despite high productivity is similar, which is that the orders are sent to the "wrong" ergo slow

machines. In this particular case, as we see in *Figure 14*, it can happen that all machines are stressed and very busy at the beginning of production, but they are still slow.

In the second plot, however, the orders are far better distributed and are only processed on the "right" (=fast) machines, which is very important with such high processing times. In the first phase, we only see red and blue orders going through the system. They are processed exclusively on M0, M2, M3, M5, M6 and M8, while M4 only appears in the second phase during yellow-order processing. M1, and M7 are omitted completely.

It should also be noted that in *Figure 14* some machine workloads exceed the 100 percent mark. Naturally, the utilization cannot exceed 100 percent. These ripples are due to the used measurement method in AnyLogic. It sometimes occurs at the transitions between two time intervals that the working times are assigned to the wrong interval. Consequently, a working time block might be assigned to a wrong measurement interval and is missing in the subsequent one. Therefore, we obtain apparent utilizations of over 100 percent at certain points. We already solve this deviation problem by plotting the utilization averages of ten experiment replications. As a result, the rippling effect is largely suppressed, but can never be eliminated entirely.

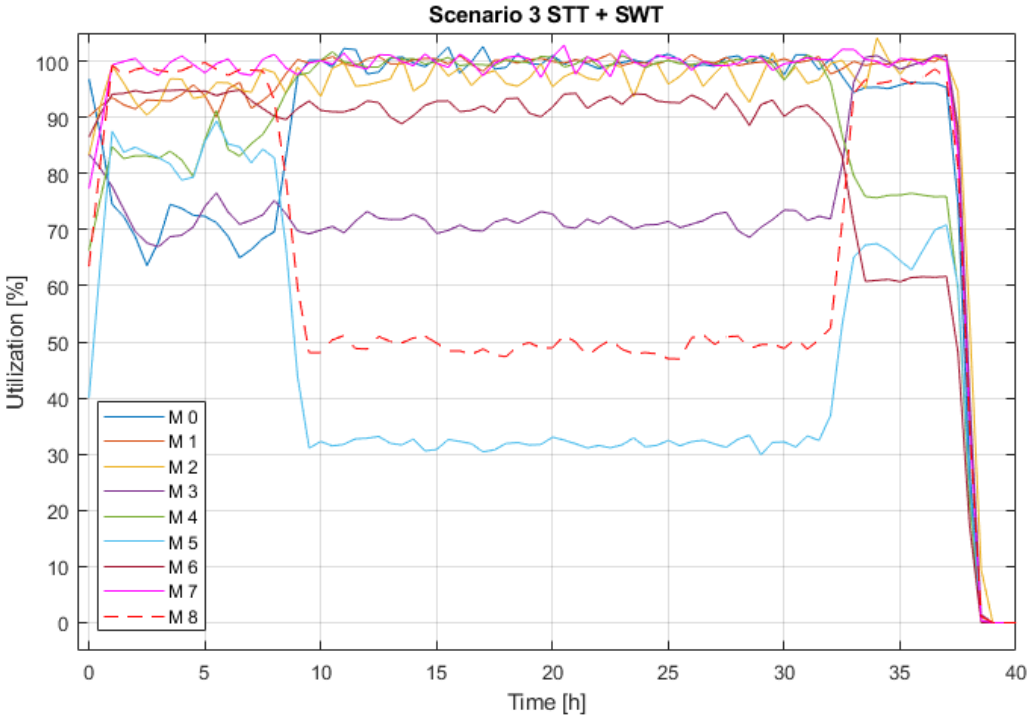

Figure 14: Productivity of machines over time I

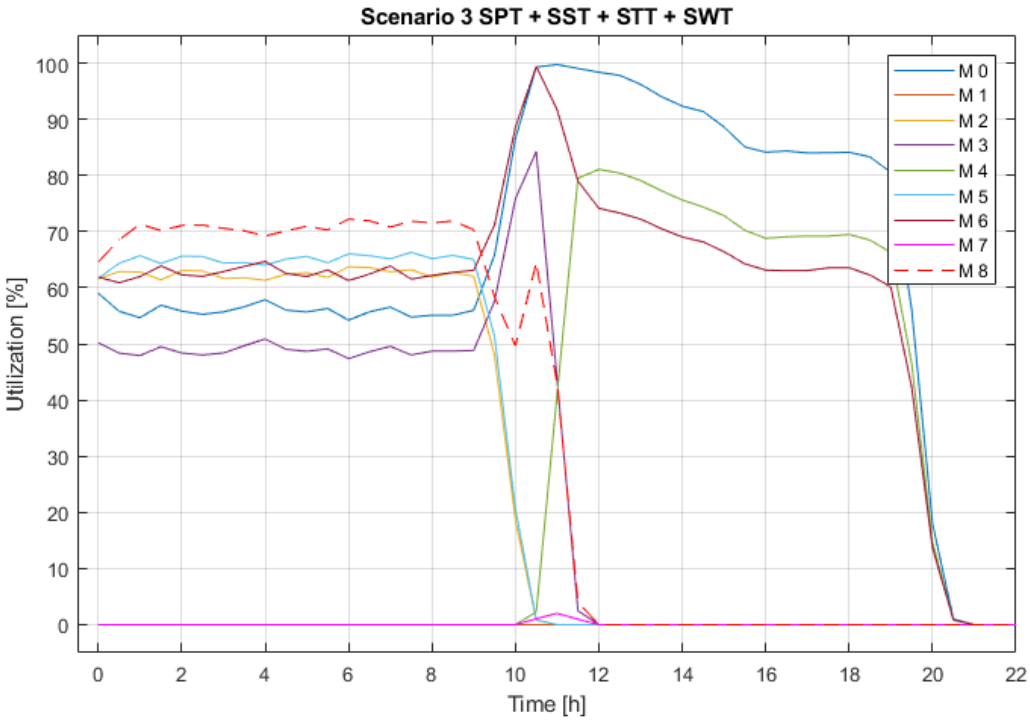

Figure 15: Productivity of machines over time II

In addition to the very intelligent machine assignment that we see in SPT+SST+STT+SWT, we would also like to argue that this rule could become the most effective due to the balance of all parameters that we have included in this policy. According to our hypothesis, although PT is very important, it is not too high to completely outweigh the other parameters, especially since we always depart from a scenario in which transport times are very high in significance. We believe that if the system is designed in a balanced way, raw materials need similar times for everything and the “all in one” rule can triumph.

5.5. Scenario 4 – Scheduling with Large ST

With respect to this rule, we can say that there are many commonalities with regard to the first scenario. It is pretty obvious which rule is going to lose when it comes to setup times, but it is also quite obvious that the rule will win which consists of a combination of first scenario's winner and the newly increased parameter (setup times). These ST values used for Scenario 4 can be found on *Table 4*, however this change was very simple: take the numbers of the setup matrix and multiply by 50.

	SQ	SPT	SST	STT	SWT	SPT+SST	SPT+STT	SST+STT	SPT+SWT	SST+SWT	STT+SWT	SPT+STT+SWT	SPT+SST+SWT	SST+STT+SWT	SPT+SST+STT	SPT+SST+STT+SWT
Red_makeSpan	107,7	17,2	7,5	7,5	6,1	11,9	8,9	7,5	46,9	6,1	35,7	68,6	5,9	4,9	7,5	5,3
Blue_makeSpan	107,8	16,6	24,3	16,7	13,0	26,0	14,4	21,4	12,5	21,1	40,7	61,2	24,2	14,9	23,5	17,0
Yellow_makeSpan	108,1	25,8	15,1	23,7	21,0	16,4	21,4	14,3	51,4	14,4	44,3	73,3	15,3	10,0	14,3	11,4
MakeSpan	**108,4**	**25,8**	**24,3**	**23,7**	**21,0**	**26,0**	**21,4**	**21,4**	**51,4**	**21,1**	**44,3**	**73,3**	**24,2**	**14,9**	**23,5**	**17,0**
M0 Utilization	0,1	0,3	1,0	0,3	0,8	0,3	0,3	0,3	0,1	0,8	0,2	0,1	0,3	0,5	0,3	0,5
M1 Utilization	0,1	0,0	0,0	0,3	0,3	0,0	0,0	0,0	0,0	0,3	0,2	0,0	0,1	0,6	0,0	0,3
M2 Utilization	0,1	0,5	0,0	0,3	0,1	0,5	0,6	0,6	0,2	0,1	0,1	0,1	0,5	0,4	0,6	0,5
M3 Utilization	0,1	0,2	0,8	0,2	0,8	0,2	0,3	0,3	0,1	0,8	0,2	0,1	0,2	0,5	0,2	0,2
M4 Utilization	0,1	0,5	0,0	0,3	0,2	0,5	0,0	0,0	0,2	0,2	0,2	0,1	0,5	0,4	0,0	0,4
M5 Utilization	0,1	0,0	0,0	0,3	0,0	0,0	0,6	0,6	0,0	0,0	0,1	0,1	0,0	0,4	0,6	0,5
M6 Utilization	0,1	0,5	0,9	0,3	0,8	0,8	0,3	0,4	0,1	0,8	0,2	0,1	0,5	0,5	0,3	0,4
M7 Utilization	0,1	0,0	0,0	0,4	0,2	0,0	0,3	0,0	0,1	0,2	0,2	0,1	0,0	0,6	0,6	0,3
M8 Utilization	0,1	0,3	0,0	0,3	0,0	0,0	0,3	0,6	0,2	0,0	0,1	0,1	0,4	0,4	0,0	0,5
Utilization	**0,1**	**0,2**	**0,3**	**0,3**	**0,3**	**0,3**	**0,3**	**0,3**	**0,1**	**0,3**	**0,2**	**0,1**	**0,3**	**0,5**	**0,3**	**0,4**
M0 Setups	678,3	0,0	2,0	0,0	2,0	0,0	0,0	0,0	1,0	2,0	2,0	1,0	2,0	2,0	0,0	2,0
M1 Setups	648,4	0,0	0,0	1,0	2,0	0,0	0,0	0,0	3,0	2,0	133,4	15,0	2,0	2,0	0,0	2,0
M2 Setups	643,9	2,0	0,0	1,0	2,0	2,0	2,0	2,0	2,0	2,0	200,9	2,0	2,0	2,0	2,0	2,0
M3 Setups	675,0	0,0	2,0	0,0	2,0	0,0	0,0	0,0	57,3	2,0	185,2	28,7	2,0	2,0	0,0	2,0
M4 Setups	653,5	2,0	0,0	1,0	2,0	2,0	0,0	0,0	119,5	2,0	186,9	49,9	2,0	2,0	0,0	2,0
M5 Setups	597,0	0,0	0,0	1,0	2,0	0,0	2,0	2,0	58,6	2,0	225,9	26,6	2,0	2,0	2,0	2,0
M6 Setups	677,7	1,0	2,0	0,0	2,0	2,0	0,0	0,0	239,5	2,0	197,5	380,9	2,0	2,0	0,0	2,0
M7 Setups	646,4	0,0	0,0	1,0	2,0	0,0	1,0	0,0	221,1	2,0	193,8	368,5	2,0	2,0	2,0	2,0
M8 Setups	574,5	1,0	0,0	1,0	2,0	0,0	1,0	2,0	223,7	2,0	223,9	379,4	2,0	2,0	0,0	2,0
No. Setups	**643,9**	**0,7**	**0,7**	**0,7**	**2,0**	**0,7**	**0,7**	**0,7**	**102,9**	**2,0**	**172,2**	**139,1**	**2,0**	**2,0**	**0,7**	**2,0**

Table 10: Results of Scenario 4

What we see here, when we compare the two lines of the table, is that the execution time series and utilization series again have a perfect "colour match" with a maximum of 50% utilization at the fastest and a minimum of 10% at the slowest policy. However, in view of the colour switches with such high values, we wanted to observe once again the differences between the times normally used by the machines and the exclusively productive times-of-use

for three rules: SQ with very high, SPT+STT+SWT with high and SST+STT+SWT with low number of setups. The results of these runs can be found in *Figure 16*:

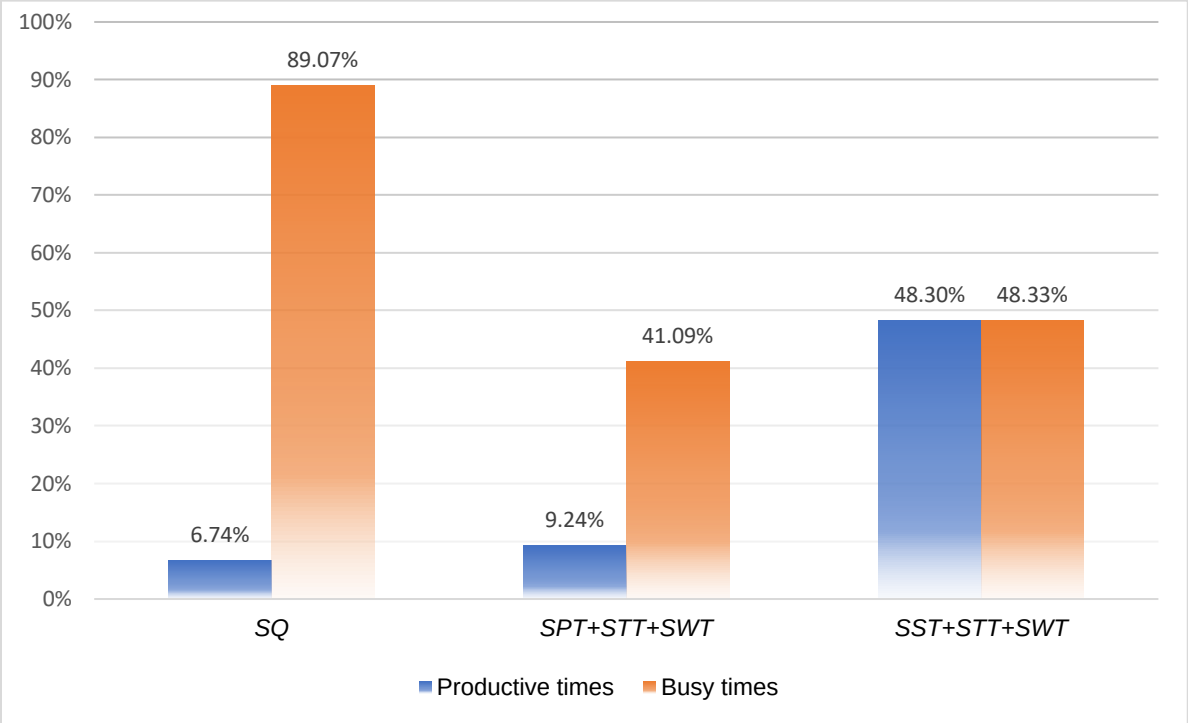

Figure 16: Busy vs Productive Utilization times within Scenario 4

Aside from the huge difference between the two types of machine usage within the SQ policy, we also noticed that there is a remarkable difference in the utilization, when we observe the SPT+STT+SWT policy, which has an average of "only" 140 colour changes during production. This difference of 40% counts as big, because if we remember Scenario 1, we had in case of the best rule with 200 setups over time only about 10% difference between "busy" and productive utilization of the machines. These 40% are therefore to be written completely on account of the increased setup times. And finally, as far as our best rule (SST+STT+SWT) is concerned, there are no real differences worth mentioning, even if we take the setups into account in the calculation of workload.

5.6. Scenario 5 – Scheduling with Two Transport Units

Just like so far, also in this scenario our starting point was Scenario 1. This is a special case, however, because all the key parameters stayed unchanged. Nevertheless, one factor changed – as the headline already indicates – that is the number of transport units. As you will see in the table, this results in that our policies also perform almost identically to Scenario 1 with the difference that thanks to the second crane, all makespans turn out to be not exactly twice as small as before, but significantly better. Also, our number one rule seems to stay the same (STT+SWT). On the contrary, however, the worst rules seem to change through this "little" modification from SPT+SST into SST+STT. Nevertheless, it is once again very obvious that just like in Scenario 1, every rule without SWT consideration performs very poorly, whereas SQ and everything else including SWT finish production below 11.5 hours. Not surprisingly, also in this setup, all policies in which STT and SWT get combined head up in front line. The two best rules stay the same with an average machine utilization of 0,8.

	SQ	SPT	SST	STT	SWT	SPT+SST	SPT+STT	SST+STT	SPT+SWT	SST+SWT	STT+SWT	SPT+STT+SWT	SPT+SST+SWT	SST+STT+SWT	SPT+SST+STT	SPT+SST+STT+SWT
Red_makeSpan	11,2	14,5	13,1	9,2	3,6	7,5	7,5	8,4	8,9	3,5	4,8	7,2	3,4	5,1	7,5	5,8
Blue_makeSpan	11,2	13,7	23,8	16,0	7,3	13,7	13,7	15,9	5,5	11,1	7,0	5,3	7,8	7,0	20,6	6,4
Yellow_makeSpar	11,2	20,6	14,6	22,8	11,1	20,5	20,5	24,6	11,4	7,5	8,8	9,3	11,2	8,8	13,7	9,2
MakeSpan	**11,3**	**20,6**	**23,8**	**22,8**	**11,1**	**20,5**	**20,5**	**24,6**	**11,4**	**11,1**	**8,8**	**9,3**	**11,2**	**8,8**	**20,6**	**9,2**
M0 Utilization	0,8	0,3	1,0	0,3	0,9	0,3	0,3	0,3	0,7	0,9	0,9	0,9	0,6	0,9	0,3	0,9
M1 Utilization	0,8	0,0	0,0	0,3	0,8	0,0	0,0	0,7	0,2	0,8	0,9	0,5	0,5	0,9	0,0	0,5
M2 Utilization	0,6	0,6	0,0	0,3	0,4	0,6	0,6	0,0	0,9	0,4	0,7	0,9	0,8	0,8	0,6	0,9
M3 Utilization	0,8	0,3	0,6	0,3	0,9	0,3	0,3	0,2	0,5	0,9	0,8	0,6	0,5	0,8	0,3	0,6
M4 Utilization	0,6	0,6	0,2	0,3	0,7	0,6	0,0	0,5	0,8	0,7	0,7	0,5	0,8	0,7	0,0	0,5
M5 Utilization	0,3	0,0	0,0	0,3	0,1	0,0	0,6	0,0	0,3	0,1	0,7	0,8	0,4	0,7	0,6	0,9
M6 Utilization	0,8	0,6	0,9	0,3	1,0	0,6	0,4	0,3	0,8	1,0	0,8	0,8	0,8	0,8	0,4	0,8
M7 Utilization	0,7	0,0	0,0	0,4	0,8	0,0	0,3	0,6	0,4	0,8	0,8	0,8	0,5	0,8	0,3	0,9
M8 Utilization	0,4	0,3	0,0	0,3	0,3	0,3	0,3	0,0	0,6	0,3	0,8	0,7	0,7	0,8	0,3	0,7
Utilization	**0,6**	**0,3**	**0,3**	**0,3**	**0,7**	**0,3**	**0,3**	**0,3**	**0,6**	**0,7**	**0,8**	**0,7**	**0,6**	**0,8**	**0,3**	**0,7**
M0 Setups	745,9	0,0	4,0	0,0	2,2	0,0	0,0	0,0	1,0	4,0	2,2	1,2	1,0	2,1	0,0	1,4
M1 Setups	669,0	0,0	0,0	1,0	2,0	0,0	0,0	2,0	3,0	2,0	254,8	42,3	2,0	2,0	0,0	2,2
M2 Setups	556,0	2,0	0,0	1,0	2,4	2,0	2,2	0,0	2,0	2,0	461,1	2,2	2,2	1,4	2,0	5,0
M3 Setups	839,8	0,0	3,0	0,0	4,4	0,0	0,0	0,0	6,4	2,2	75,5	6,6	3,4	22,3	0,0	4,7
M4 Setups	772,1	2,0	0,0	1,0	4,4	2,0	0,0	2,0	103,9	3,8	267,5	96,8	9,7	11,6	0,0	54,5
M5 Setups	360,4	0,0	0,0	1,0	2,2	0,0	2,2	0,0	4,1	2,0	436,5	2,4	9,7	5,8	2,0	4,6
M6 Setups	810,5	2,6	3,0	0,0	5,2	1,0	0,0	0,0	535,0	2,8	319,2	381,9	55,9	299,6	0,0	266,6
M7 Setups	680,7	0,0	0,0	1,0	4,6	0,0	1,0	2,2	160,1	2,8	296,2	360,3	16,7	271,1	1,0	245,1
M8 Setups	454,8	1,0	0,0	1,0	2,4	1,0	1,0	0,0	229,4	2,4	406,9	388,3	1,3	317,6	1,0	276,3
No. Setups	**654,4**	**0,8**	**1,1**	**0,7**	**3,3**	**0,7**	**0,7**	**0,7**	**116,1**	**2,7**	**280,0**	**142,4**	**11,3**	**103,7**	**0,7**	**95,6**

Table 11: Results of Scenario 5

	SQ	SPT	SST	STT	SWT	SPT+SST	SPT+STT	SST+STT	SPT+SWT	SST+SWT	STT+SWT	SPT+STT+SWT	SPT+SST+SWT	SST+STT+SWT	SPT+SST+STT	SPT+SST+STT+SWT
S1 - MakeSpan	20,4	25,1	23,9	22,8	20,4	25,1	20,6	23,2	23,0	20,5	13,8	15,9	23,6	14,5	20,6	15,9
S5 - MakeSpan	11,3	20,6	23,8	22,8	11,1	20,5	20,5	24,6	11,4	11,1	8,8	9,3	11,2	8,8	20,6	9,2
S1 - Utilization	0,4	0,3	0,3	0,3	0,4	0,3	0,3	0,3	0,3	0,4	0,5	0,4	0,3	0,5	0,3	0,4
S5 - Utilization	0,6	0,3	0,3	0,3	0,7	0,3	0,3	0,3	0,6	0,7	0,8	0,7	0,6	0,8	0,3	0,7
S1 - No. Setups	657,9	0,7	0,7	0,7	2,7	0,7	0,7	1,0	59,4	1,8	204,9	34,2	1,4	6,0	1,4	4,2
S5 - No. Setups	654,4	0,8	1,1	0,7	3,3	0,7	0,7	0,7	116,1	2,7	280,0	142,4	11,3	103,7	0,7	95,6

Table 12: Results of Scenario 1 & 5 in comparison

We would like to dedicate a few words on the implementation of the 2-crane scenario here. First, we want to specify that there are many different ways to create a second transport unit in the program. Depending on how you want to define the second unit, you may create a new agent and extend the model with a second resource pool and a second service between the stages or you can simply add another resource unit to the existing resource pool as well. Since we did not want to define any new properties for the second crane for this scenario, we have extended the existing resource pool by one and simply "copied" the properties of the existing crane.

Nevertheless, we changed the criteria after the jobs are delivered (= job sequencing), since we can now even deliver two jobs simultaneously using two cranes instead of one. In order for this to work properly, we have also had to give each order with the second highest priority the state "true". We did this by removing the job with the highest priority from our ArrayList and re-evaluating the remaining list to be able to filter out the second highest priority. This means that once the capacity of the resource pool now consisting of two cranes is at least one (i.e. at least one crane is free), the job with the second highest index can be given the free "pass".

What really interested us in this scenario, apart from the question of whether the performance of the rules has changed, was the question of whether it was worth hiring a second crane to deliver the orders. Although the solution for the uncertainty about the performance of the rules with the examined transport units was not necessarily intuitive, it is actually quite reasonable when one considers that nothing has changed fundamentally in the model, except the transport capacity. What remains, is the question, why, compared to the initial scenario,

production can in some cases be accelerated twice as fast as before and in some cases it does not improve at all (see *Table 12* for comparison). According to our interpretation, this depends on where the bottleneck lies in the system. If the machines are fast and jobs frequently have to "wait" for the crane, an additional crane will significantly improve the overall system - in this case the bottleneck is the crane. In the other case however, if the machines are comparatively slow and the crane is not working at full capacity, an additional crane will not improve the makespan. So, the bottleneck lies with the machines. A good example of the former is SPT+SWT or SPT+SST+SWT (one of the worst rules so far), which is twice as fast thanks to the second crane in this case. STT and SST rules, however, could not increase performance because they promote uniform machine selection, i.e. orders are usually sent to the same machines, thus a second crane does not improve the process.

Based on these findings, it can be argued that for these two rules and for others that contain SPT and SWT at the same time, it can be quite profitable to install a second crane, as production can show a very significant improvement in overall production time. In other cases, however, where the bottleneck is not the transport unit and therefore no advantages can be achieved, it naturally does not pay to use a second crane.

There is a second interpretation, which could of course also be used to explain our results, namely that due to the second transport unit the selection of the next order or machine may be subject to some errors. This is simply because the machine and job selection takes place as soon as a new job reaches a Hold element and the conditions for this new job are recalculated, regardless of whether the previous order is still in delivery or has already arrived on the target machine. Nor should we forget, if we already point out these uncertainties, that our "congestion prevention" tool against blocking, which we briefly mentioned in the implementation section, can also lead to sub-optimal workflows, since not only the jobs that have the highest priority will be forwarded, but also those that could cause a blockage in the system at a certain point in time.

5.7. Summary of the findings

In this section we would like to summarize our most important findings. During the runs and testing and evaluation we often came across interesting surprises, but we managed to decipher almost all details of the program with thoughtful work.

Just as the performance evaluation was divided into three categories, we would also like to divide the findings into these three categories: makespan, machine utilization and number of setups:

1) We have noticed that the performance of the rules depends to a large extent on how dominant the individual parameters of the rule are. Although we would have expected that the overall production times are always best estimated with the most sophisticated rule and that it will defeat other rules most of the time, we had to realize that sometimes it was smarter to use simpler rules and ignore some parameters. Namely, taking into account some "unimportant" parameters, even if they were already weighted by adding the respective times, the entire completion time could be slowed down. Furthermore, we also noticed that the initial scenario (scenario 1) was very present during all other parameter variation runs, although we changed some parameters. The winner of the initial scenario, STT+SWT, was always part of the winning rule in all other scenarios except Scenario 2. This let us definitely conclude on its important role.

However, what was even more exciting for us to realize was the fact that our rainbow policy (SPT+SST+STT+SWT) was proved to be the best rule in comparison across all scenarios, after examining the performance in connection with total completion time for all rules in *Table 13*. This outcome really surprised us because this particular policy could actually only win once out of 5 scenarios (in Scenario 3) and otherwise was always considered somewhere between the second to fourth best rule. We can now see from *Table 13* that the average speed of this rule over all scenarios is only 20 hours, making it the best of all. The next best rule over the system is SST+STT+SWT, the third best SST+SWT and the fourth and fifth best SWT and SPT+STT. Although these rules have never/nearly never been the winners of the individual scenarios; generally speaking, they seem to be very well applicable.

2) Next, we discovered that, with one exception, we can claim that the rules that led to the smallest makespan, that is, the fastest rules, gave us the highest productivity. However, in Scenario 3, we have discovered that the fact that the machines work and are highly "productive" over time does not always mean that they are fast! Especially when the machines of a processing stage perform very differently, for example because of their specialized knowledge for a certain job, it is very important, independent of productivity, that these jobs are assigned to the right machines. We can therefore draw the conclusion from this finding that although productivity is indeed connected with the total completion time, it does not always mean that the policy with the highest utilization leads to the smallest makespan. Therefore, maximizing utilization is not always the best way to minimize $E(C_{max})$.

Nevertheless, if we look at the 5 best performing rules mentioned earlier, it is not wrong to state that they can be associated with a fairly good productivity compared to other policies.

3) As far as the number of setups is concerned, we have also gathered significant new knowledge. Let us take for example the rules with the shortest transport times. Although the model wanted to depict a production by storing the orders in a large "pool" on the raw material level and in which the jobs would have been taken out of this initial lucky bag, where all types of jobs have the same chances of being elected, it has developed differently through the way of implementation and intelligence of the program. So, after we had different starting points for the different colours, this rule (STT) selected firstly the colours that were closest to the individual machines and sent them off first. Then, when we had increased the transport times in Scenario 2, this distance from the different types to machines became even more relevant, and once the distribution function (SWT) had interacted, the results for the total completion time worsened. Aside from this insight, which explains most of the time the very small number of setups, we have also found that this factor (number of setups), as long as it is kept small, is as good as negligible in relation to so many (3000) jobs.

On the general picture, what we can highlight here is that the best 5 rules have a very low number of setups considering all scenarios. If we look at the rules with a rather large number of setups, we can see that SQ ends up only at the 13^{th} place out of 16 in the ranking and the policy with the second highest number of setups (STT+SWT) in line with the SPT+STT+SWT rule ends up somewhere in the "upper middle class" of rules. Although, the

former rule (STT+SWT) could "win" in two scenarios, it has generally not performed too well due to its polarizing character and fairly high number of setups.

	SQ	SPT	SST	STT	SWT	SPT+SST	SPT+STT	SST+STT	SPT+SWT	SST+SWT	STT+SWT	SPT+STT+SWT	SPT+SST+SWT	SST+STT+SWT	SPT+SST+STT	SPT+SST+STT+SWT
S1 - MakeSpan	20,4	25,1	23,9	22,8	20,4	25,1	20,6	23,2	23,0	20,5	13,8	15,9	23,6	14,5	20,6	15,9
S2 - MakeSpan	44,3	53,7	43,9	33,2	44,1	53,7	34,5	34,3	53,7	44,1	34,8	36,3	53,1	35,5	34,0	38,3
S3 - MakeSpan	55,4	22,4	171,5	234,4	20,9	22,4	21,7	226,0	22,2	21,0	37,8	19,7	22,2	37,1	21,4	19,6
S4 - MakeSpan	108,4	25,8	24,3	23,7	21,0	26,0	21,4	21,4	51,4	21,1	44,3	73,3	24,2	14,9	23,5	17,0
S5 - MakeSpan	11,3	20,6	23,8	22,8	11,1	20,5	20,5	24,6	11,4	11,1	8,8	9,3	11,2	8,8	20,6	9,2
Mean	**48,0**	**29,5**	**57,5**	**67,4**	**23,5**	**29,5**	**23,8**	**65,9**	**32,3**	**23,6**	**27,9**	**30,9**	**26,9**	**22,2**	**24,0**	**20,0**
S1 - Utilization	0,4	0,3	0,3	0,3	0,4	0,3	0,3	0,3	0,3	0,4	0,5	0,4	0,3	0,5	0,3	0,4
S2 - Utilization	0,2	0,1	0,2	0,2	0,2	0,1	0,2	0,2	0,1	0,2	0,2	0,2	0,1	0,2	0,2	0,2
S3 - Utilization	0,8	0,3	0,2	0,2	0,8	0,3	0,3	0,2	0,3	0,8	0,8	0,3	0,3	0,8	0,3	0,3
S4 - Utilization	0,1	0,2	0,3	0,3	0,3	0,3	0,3	0,3	0,1	0,3	0,2	0,1	0,3	0,5	0,3	0,4
S5 - Utilization	0,6	0,3	0,3	0,3	0,7	0,3	0,3	0,3	0,6	0,7	0,8	0,7	0,6	0,8	0,3	0,7
Mean	**0,4**	**0,2**	**0,3**	**0,3**	**0,5**	**0,2**	**0,3**	**0,3**	**0,3**	**0,5**	**0,5**	**0,4**	**0,3**	**0,6**	**0,3**	**0,4**
S1 - No. Setups	657,9	0,7	0,7	0,7	2,7	0,7	0,7	1,0	59,4	1,8	204,9	34,2	1,4	6,0	1,4	4,2
S2 - No. Setups	658,9	0,7	0,7	0,7	1,3	0,7	157,7	0,7	1,1	1,0	44,1	287,3	0,8	19,1	2,1	2,1
S3 - No. Setups	661,1	0,7	0,7	0,7	3,1	0,7	0,7	0,8	1,1	2,9	159,9	1,1	1,0	11,7	0,7	1,0
S4 - No. Setups	643,9	0,7	0,7	0,7	2,0	0,7	0,7	0,7	102,9	2,0	172,2	139,1	2,0	2,0	0,7	2,0
S5 - No. Setups	654,4	0,8	1,1	0,7	3,3	0,7	0,7	0,7	116,1	2,7	280,0	142,4	11,3	103,7	0,7	95,6
Mean	**655,2**	**0,7**	**0,8**	**0,7**	**2,5**	**0,7**	**32,1**	**0,8**	**56,1**	**2,1**	**172,2**	**120,8**	**3,3**	**28,5**	**1,1**	**21,0**

Table 13: Summary of all Scenarios

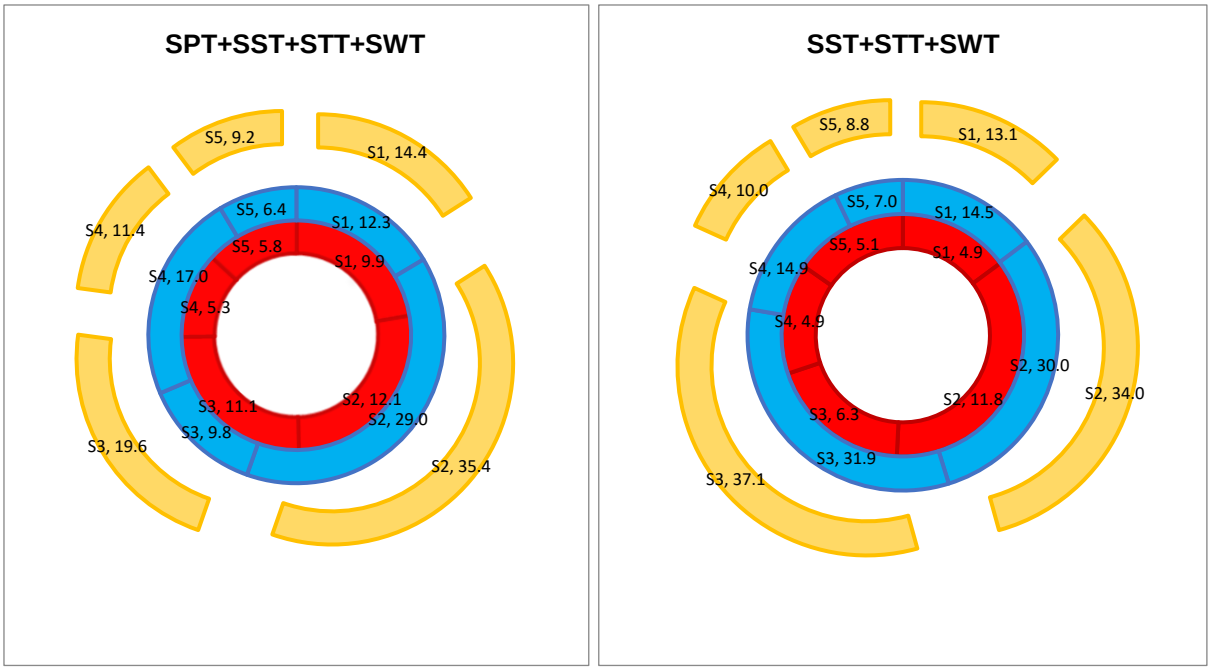

Figure 17: Total completion times according to types and to scenarios

While what we can see on *Figure 17* is certainly interesting, one should not draw any concrete conclusions from it. The two best general rules can be seen here at a glance with a breakdown of the three types of orders over the five scenarios. In general, it can be said that under the influence of these rules, red jobs are produced most of the time first, then blue and then yellow with some exceptional scenarios. But as we said, this information does not seem too meaningful, since when we look at the worst performing rule, we can actually see the same trend. We therefore believe that it has much more to do with our settings, e.g. which colour we have defined as "initial" on the machines, namely red.

6. Conclusion

One of the major problems in flexible flow shop manufacturing has been to dispatch different types of jobs among a varying number of machines per production stage while trying to reach the optimum outcome in different aspects. Therefore, a special model has been designed for this thesis which can change its input data flexibly and makes the effect of different disposition rules and priority policies accessible for future considerations, taking various scenarios into account.

After a long programming period and many adjustments of the program to our purposes, we could not only understand how the system works, but also how the problem works. We then came to the conclusion that the performance of the rules largely depends on the dominance of the individual parameters in individual scenarios and that there is no best rule, but that the performance of the respective policy under uncertainties depends only on the experimental setup. Further studies could include different threshold and sensitivity analyses of the desired input data, in which the role of the individual parameters could be represented even better and the right logic could be selected for each additional individual scenario. However, if it is clear that the individual parameters can change due to uncertainties, and if we are looking for a truly universal logic, it is a good approach to choose a rule that does not come too close to a particular case. As we have already quoted in our 'Motivation' section, *"Algorithms must be able to find solutions which remain robust under different scenarios"* (Ruiz & Vázquez-Rodríguez, 2010). Of all the policies, therefore, choosing a logic that takes everything into consideration (SPT+SST+STT+SWT) seems to us to be the best strategy.

We hope that not only this thesis but also our source code, which was written for the simulation program, can serve a good purpose and that more research can continue to expand this commitment in the future.

Bibliography

Akrami, B. K. (2006). Two metaheuristic methods for the common cycle economic lot sizing and scheduling in flexible flow shops with limited intermediate buffers: the finite horizon case. *Appl Math Comput, 183*(1), 634–645.

Allahverdi, A., Ng, C. T., Cheng, T. E., & Kovalyov, M. Y. (2008). A Survey of Scheduling Problems with Setup Times. European Journal of Operational Research, 187, 985–1032.

AnyLogic. (2018). *AnyLogic Help*. Retrieved July 20, 2018, from https://help.anylogic.com/index.jsp

Baker, K. R. (1975). A comparative study of flow-shop algorithms. Operations Research, 23(1), 62-73.

Benda, F., Braune, R., Doerner, K. F., & Hartl, R. F. (2018). A machine learning approach for flow shop scheduling problems with alternative resources, sequence-dependent setup times, and blocking. Vienna: University of Vienna.

Blazewicz, J. e. (2007). Handbook on scheduling: from theory to applications. Springer Science & Business Media.

Cambridge Dictionary. (2018). Retrieved May 25, 2018, from http://dictionary.cambridge.org

Gondek, V. (2011). Hybrid Flow-Shop Scheduling mit verschiedenen Restriktionen: Heuristische Lösung und LP-basierte untere Schranken . Viersen: Universität Duisburg-Essen.

González-Neira, E. M., Montoya-Torres, J. R., & Barrera, D. (2017, January). Flow-shop scheduling problem under uncertainties: Review and trends. *International Journal of Industrial Engineering Computations 8(4)*, pp. 399-426.

Gourgand, M., Grangeon, N., & Norre, S. (2003). A contribution to the stochastic flow shop scheduling problem. *European Journal of Operational Research*(151), 415–433.

Gupta, J. (1988). Two-stage, hybrid flow shop scheduling problem. *Journal of the Operational Research Society , 39* (4), 359–364.

Kadipasaoglu, S., Xiang, W., & Khumawala, B. (1997). A comparison of sequencing rules in static and dynamic hybrid flow systems. *International Journal of Production Research, 35*(5), 1359–1384.

Linn, R., & Zhang, W. (1999). Hybrid flow shop scheduling: A survey. *Computers & Industrial Engineering, 37*, 57 – 61.

Ouelhadj, D., & Petrovic, S. (2009). A survey of dynamic scheduling in manufacturing systems. *Journal of scheduling, - Springer, 4*(12), 417-544.

Oxford Learner's Dictionary. (2018). Retrieved May 25, 2018, from http://www.oup.com/elt/catalogue/teachersites/oald7/lookup?cc=global

Pegden, C. D., Shannon, R. E., & Sadowski, R. P. (1995). *Introduction to simulation using SIMAN* (Vol. 2. ed.). New York: McGraw-Hill.

Pinedo, M. L. (2016). Flow shops and flexible flow shops (Deterministic). In *Scheduling* (pp. 151-181). Springer, Cham.

Pinedo, M. L. (2016). Flow Shops, Job Shops and Open Shops (Stochastic). In *Scheduling* (pp. 349-372). Springer, Cham.

Quadt, D., & Kuhn, H. (2007). A taxonomy of flexible flow line scheduling procedures. In *European Journal of Operational Research 178 (), Nr. 3* (pp. 686 – 698).

Rajendran, C., & Holthaus, O. (1999). A comparative study of dispatching rules in dynamic flowshops and jobshops. *European journal of operational research, 116(1)*, pp. 156-170.

Ramasesh, R. (1990). Dynamic job shop scheduling: a survey of simulation research . *Omega, 18(1)*, pp. 43-57.

Ribas, I., Leisten, R., & Framin, J. M. (2010). Review and classification of hybrid flow shop scheduling problems from a production system and a solutions procedure perspective. *Computers & Operations Research*(37), 1439–1454.

Rossi, A., & Dini, G. (2007). Flexible job-shop scheduling with routing flexibility and separable setup times using ant colony optimisation method . *Robotics and Computer-Integrated Manufacturing*(23), 503–516.

Ruiz, R., & Vázquez-Rodríguez, J. A. (2010). The hybrid flow shop scheduling problem. *European Journal of Operational Research*(205), 1 – 18.

Sabuncuoglu, I. (1998). A study of scheduling rules of flexible manufacturing systems: a simulation approach. *International Journal of Production Research, 36(2)*, pp. 527-546.

Salvador, M. (1973). A solution to a special case of flow shop scheduling problems. In S. Elmaghraby (Ed.), *Symposium of the Theory of Scheduling and Applications* (pp. 83 – 91). Springer.

Sawik, T. (2000). Mixed integer programming for scheduling flexible flow lines with limited intermediate buffers. *Math Comput Model, 31*(13), 39–52.

Wittrock, R. (1988). An adaptable scheduling algorithm for flexible flow lines. *Operations Research, 36*(3), 445 – 453.

Xuan, H., & Tang, L. (2007). Scheduling a hybrid flowshop with batch production at the last stage. *Computers & Operations Research, 34*(5), 2718 – 2733.