The Autonomous Batch Scheduler

Albers, M.P.

Award date:
2022

Department of Mechanical Engineering
Manufacturing Systems Engineering
Dynamics & Control

The Autonomous Batch Scheduler

Master Graduation Thesis

Student name:	M.P. Albers
Student ID:	0893475
Research group:	Dynamics & Control
Master program:	Manufacturing Systems Engineering in Mechanical Engineering
Supervisors TU/e	Prof. dr. ir. I.J.B.F. Adan dr A.E. Akcay dr. ir. A.A.J. Lefeber
External Supervisors	J. Adan (Nexperia) J.P. Driessen (Nexperia)
DC Number	DC 2022.006

This report is made in accordance with the TU/e Code of Scientific Conduct for the Master thesis

Eindhoven, Wednesday 26th January, 2022

Abstract

This thesis has investigated the use of a closed-loop control system to solve scheduling problems in the semiconductor manufacturing environment. This research is part of the Batch Scheduler project, that uses a hybrid genetic algorithm to solve the resource allocation problem. By developing a simulation that mimics the real-world behaviour, the closed-loop manufacturing environment has been tested and evaluated on the performances. With a focus on the back-end production of integrated circuits, this research addresses an unrelated parallel machine scheduling problem with sequence- and machine dependent setup times, machine eligibility and due dates.

The scope of the simulation takes 35 assembly lines into consideration with a total of 109 machines. The simulation uses a weekly arrival rate, with a new schedule generated at the start of each week. Based on the data acquired from the simulation, three unique rescheduling heuristics have been developed. One of those heuristics is periodic driven and can only execute full rescheduling procedures. The other two heuristics are event driven and can execute full and partial rescheduling.

The factory performance is mainly evaluated on the throughput and tardiness performance. The use of any rescheduling approach improves the factory performances significantly. It is concluded that full rescheduling performs better with respect to throughput performance. Partial rescheduling performs better with respect to tardiness performance.

Preface

At this point in time, I am grateful and happy to present the final results of my graduation project as part of my master Manufacturing System Engineering at the Eindhoven University of Technology. After graduating for my bachelor in Mechanical Engineering in 2019, I started my master, looking forward to the international internship and graduation project. Unfortunately, after finishing my courses, COVID-19 prevented the opportunity to go abroad, requiring me to perform my internship in the Netherlands. After my internship, I was provided with the opportunity to perform my graduation project at Nexperia. Even though, COVID-19 prevented me from visiting the company or the factory abroad, I learned a lot from working for Nexperia.

First of all, I would like to thank my university supervisor Ivo Adan. You helped me find the companies to perform my internship and graduation project, which I am grateful for. You always helped me throughout my project, providing constructive feedback and giving me new insights to work on. Even though you were not initially my mentor, you stepped in when I needed one and for that I will always be thankful.

Secondly, I would like to thank Jelle Adan. My research is part of your PhD project and you were always able to guide me throughout the entire process. Building forwards on your research allowed me to achieve the results explained in this report. Your knowledge on the entire system and process has helped me a lot.

At last, I would like to thank my colleagues at Nexperia, especially Jan Driessen. Even though we barely met in person due to COVID-19, you were always able to provide feedback on my project and help me forwards whenever I got obstructed.

Marc Albers

Contents

List of Figures

List of Tables

Chapter 1

Introduction

In today's society, a world without electronic devices would be unthinkable. According to Statista [Statista, 2020], there are about 21.5 billion interconnected devices in the world. All of these devices require integrated circuits in order to operate, communicate and interact with the environment. All of these integrated circuits are produced in the semiconductor industry. The birth of semiconductors can be traced back to the invention of the rectifier (AC-DC converter) in 1874. Approximately, one century later in 1959, the bipolar integrated circuit (IC) was invented [Hitachi, 2020]. Since then, the semiconductor market has increased to a worth of over 500 billion USD and is expected to reach a worth over 720 billion USD by 2027 [Insights, 2020]. The market will continue to grow, driven by factors as increasing number of data centers, the use of advanced technologies and the growing popularity of Internet of Things (IoT) devices and smart homes. In order to keep up with the high demand, semiconductor manufacturers need to optimize their processes in order to reach the maximum output of their facilities.

Nexperia is a global semiconductor manufacturer, taking up 14% of the global market by producing more than 90 billion products annually [Geurtsen and Adan, 2020]. The production process of manufacturing ICs is complex and involves many steps. This process is divided into two main parts: the front-end process and the back-end process. The front-end process covers the steps which involve the formation of transistors directly onto the silicon wafers. The process starts out with clean wafers and ends with dies fabricated in large amounts on each wafer. These wafers are processed within the back-end to fabricate the ICs. The back-end process involves assembling and testing the ICs. A brief diagram of the process is shown in 1.1.

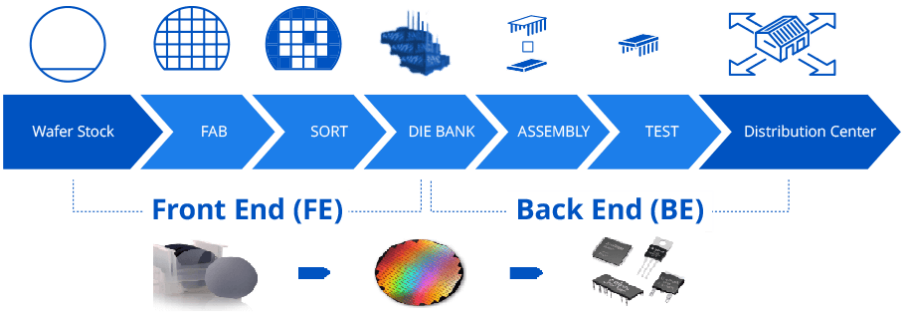


Figure 1.1: The Semiconductor manufacturing process

1.1 Assembly Process

Where the front-end facilities focus on the manufacturing of integrated circuits (so-called dies) onto bare wafers, the back-end factories focus on the production of encapsulated devices. Within the back-end facility, semi-finished ICs are encapsulated in a supporting case to prevent physical

damage and corrosion. This procedure is performed on assembly lines. Such an assembly line consists of three different processes, each performed on its own workstation.

The first operation in the assembly line is die bonding, which is the bonding of dies to the support structure, the so-called lead frame. This procedure is performed on the die bonder machine, which uses wafers as an input to pick out the dies. A die is a small block of semiconducting material on which a certain IC is fabricated. Based on the product specifications, one or more dies are placed on the lead frame. The second operation is wire bonding, which involves making the interconnections between the die and specific locations on the lead frame which will serve as connections to the outside world of the final product. The third and last procedure is molding, which is executed by the multi-plunger. Molding is encapsulating the device by a molding compound to protect the IC from potential harmful influences from the environment (e.g. moisture, particles).

These processes are performed in series on an assembly line. These assembly lines are called BIM-lines (Breakthrough In Manufacturing) and are capable of performing all three operations sequentially. In order to do so, each assembly line requires at least one die bonder workstation, one wire bonder workstation and one multi-plunger. To match the manufacturing speeds of the three consecutive steps, most assembly lines contain multiple die bonders and wire bonders, varying between one and four workstations in a row. Meanwhile, each assembly line contains only one multi-plunger to operate the molding. This is due to the fact that the die bonders and wire bonders process single products in series, while the multi-plunger processes large batches at once. In Figure 1.2, an example of a BIM-line is provided, in this case containing two die bonders, four wire bonders and one multi-plunger.

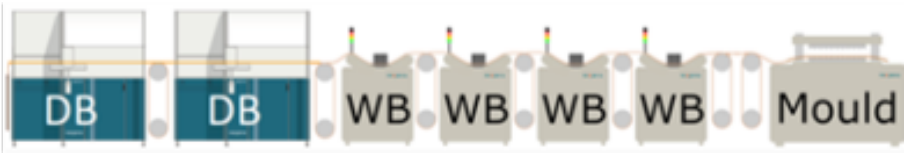


Figure 1.2: Nexperia Assembly Line

Nexperia has a total of two front-end facilities and three back-end facilities. One of the back-end facilities is the Assembly and Test facility Seremban in Malaysia (ATSN), which is the main focus of this thesis. This facility contains almost 100 assembly lines, each with varying number of workstations. Together with the other two back-end facilities, an annual production of 90 billion products is achieved, with a portfolio of 15.000 products [Nexperia, 2020]. In order to achieve these large numbers, all production processes have to be optimized. Production scheduling is an effective method to optimize the production processes, especially if no extra facilities can be bought or upgraded.

1.2 Production Scheduling

Production scheduling consists of the activities performed in a manufacturing company in order to manage and control the execution of a production process. A schedule is an assignment problem that describes into details which activities must be performed and how the factory's resources should be utilized to satisfy the plan. Production scheduling is one of the most active areas of operational research, which is largely due to the rich variety of problems within the field. These researches mainly involve parallel machine scheduling problems using ideal environments and therefore neglecting the stochastic influences. These environments typically assume a fixed number of jobs, each having its own parameters and constant machine capacities with no unexpected events or machine breakdowns. This is unrealistic and causes a gap between the theory and application of developed methods, since most of the methods do not cover real-world manufacturing environments and the large variety of influential parameters, like stochastic arrival rates, process times and

down times. With a focus on the back-end production of integrated circuits (ICs) at the Assembly and Testing facility of Nexperia, this research addresses an unrelated parallel machine scheduling problem with sequence- and machine-dependent setup times, machine eligibility and due dates.

The scheduling within ATSN was previously performed as a human exercise by a team of expert planners. They would look into all orders arriving and distribute them over the available resources in order to minimize tardiness and maximize the throughput. This human exercise has recently been replaced by a smart solution, which makes use of a Hybrid Genetic Algorithm, developed at the Research Facility in Nijmegen, Netherlands [Adan et al., 2018]. This algorithm is able to generate stable high-quality solutions and provides schedules on a weekly basis and is part of the so called Batch Scheduler project.

Chapter 2

Problem Definition

In the previous chapter, a brief introduction on the subject of this graduation project is provided. Within this chapter, a closer look at the problem at hand within Nexperia is taken. Based on this problem definition, one main research question is derived and several sub-research questions are defined to help solve the problem.

2.1 Problem definition

As mentioned in the previous chapter, the importance of optimisation within semiconductor manufacturing is critical to keep up with the high demands. Since expanding factory resources is not always an option, optimizing the scheduling approach of a factory can provide the needed increase in production volume. Over the past few years, a large project has been initiated at Nexperia, called Smart Network for optimal Planning (SNAP). Smart networks are computing networks operating with intelligence, making it possible to detect and react to events based on protocols. The intelligence is usually based on deep learning algorithms for predictive identification. If the smart network would successfully be applied to solve production scheduling problems, the system becomes autonomous. In this case, an autonomous system is a self-driven system, using the information provided to make decisions on its own, without the need of human interference.

The first step within SNAP has been achieved with the implementation of the Batch Scheduler Hybrid Genetic Algorithm (HGA). The entire SNAP is visualized in Figure 2.1, which represents a closed-loop system. A closed-loop system uses feedback of the output in order to adapt the control action and achieve the desired output. An open-loop system has no control action or a control action independent of the output. An open-loop system lacks a feedback mechanism in case the realized production schedule deviates from the optimized scenario.

Currently, the system is an open-loop system, shown by the top line in Figure 2.1. Arriving orders are sent into the scheduler, providing a weekly production schedule for the factory, which is dispatched and results in an output of finished good. In order to close the loop and create an autonomous system, two main steps have to be solved. First of all, feedback has to be derived based on the performance and output of the factory. Secondly, a control system has to be defined which uses the feedback to initiate actions.

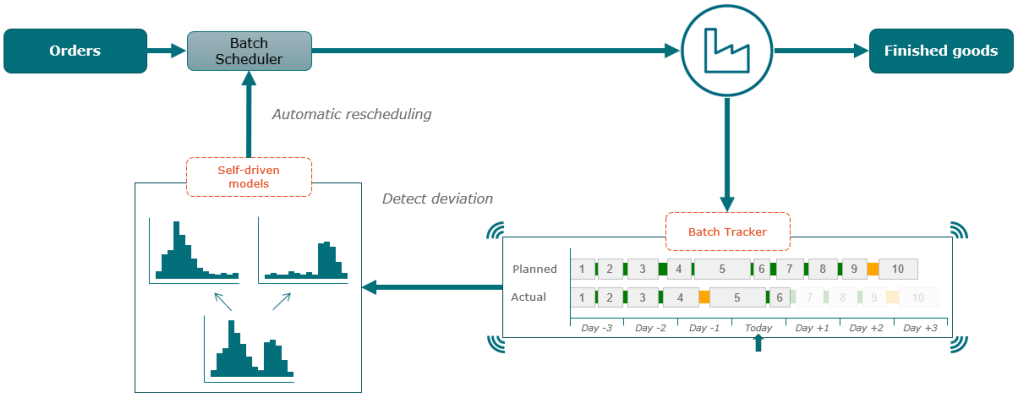


Figure 2.1: SNAP: Smart Network for optimal Planning

With the implementation of the HGA, schedule qualities should improve compared to manual derived schedules, resulting in a better efficiency of the factory. However, the only way to determine if the efficiency has been improved, is by analyzing the performance. By looking into the outputs and durations of each order and combining it with the product specifications, the throughput rates for each workstation can be determined. Based on these rates and information, the performances can be evaluated. In order to do so, data is required about the production flow.

Approximately two decades ago, all workstations within the facility have been connected to the Advanced Warning And data Collection System (AWACS). The AWACS allows a user to download Equipment State Monitoring (ESM) files, which provide data on one specific workstation. This data includes state changes, errors, amounts and other signals, all associated with time stamps. By using this data, it becomes possible to track and analyse the behaviour and performances of the workstations within the assembly lines. The data can also be used to mathematically model and predict the schedule performance, and indicate whether actions can be taken when the schedule seems to deviate too much. However, the quality of the available data is poor and currently unusable. It requires preparation in the form of filtering and cleaning, in order to create such a tracking database. With this database, the so-called Batch Tracker can be developed, allowing real-time data to be transformed to track and analyse what is going on in the factory. By creating and developing the Batch Tracker, the second step within SNAP would be solved.

Finally, a smart control system has to be developed, which uses the data to initiate actions based on triggered events. By using the real-time data as feedback for the system to make decisions and improve processes, the system becomes a closed-loop system. This research will address the problem of creating a closed-loop system, with rescheduling as the resulting event.

2.2 Research Question

The goal of the research is to upgrade the current implemented scheduling solution from an open-loop system to a closed-loop system. With the closed-loop system, feedback will be used in order to optimize the output. Furthermore, insights about rescheduling are to be obtained and the effects on performances of the factory. In order to achieve this, one main research question is derived and five sub-research questions.

Main Research Question

- How to create an autonomous Batch Scheduling system?

Sub-research questions

- How reliable is the current system?

- How to implement a rescheduling approach?
- What triggers rescheduling?
- How to prevent the need to reschedule?
- How to decide if the schedule should be partly or fully rescheduled?

Chapter 3

Methodology

This chapter describes the methodology obtained by the literature study. The chapter starts with evaluating different scheduling approaches, followed by more specific details on rescheduling approaches. Finally, scheduling in semiconductor industry will be discussed. The knowledge gained from the literature study will be used to develop rescheduling heuristics and to analyse and evaluate the performance data.

3.1 Scheduling approaches

As mentioned in Chapter 2, creating an autonomous scheduling system is the main goal of this research. With this autonomous scheduling system, an optimal solution for the scheduling problem is sought. To optimize the scheduling system, it is important to first understand the effects and influences of different approaches of scheduling in a stochastic environment. Typical optimization criteria are make span, workload, number of tardy activities and tardiness or allocation costs. Within any scheduling approach, two main elements are always involved: schedule generation and a control module. Schedule generation is the predictive planning module that determines the start and completion time of activities. The control module executes the reactive part, which involves when and how to reschedule and how to anticipate to unexpected events [Sabuncuoglu and Kizilisik, 2010].

Research has been executed in order to try and determine the best scheduling approaches for dynamical problems within a stochastic environment. Probably one of the first papers on reactive scheduling is done by Nelson and Holloway in 1977, who propose a scheduling system for a job shop with due dates and intermittent job arrivals [Holloway et al., 1977]. Their scheduling algorithm is based on the multi-pass heuristic that generates schedules at each intervention, allowing a response to unexpected events. Whereas single-pass heuristics use one method and one priority rule, multi-pass heuristics combine multiple priorities into one heuristic [Tormos and Lova, 2003]. This new way of thinking opened up opportunities for other researchers to develop new approaches for scheduling problems. By generating and reacting in different time periods, new techniques to schedule could be derived. In order to classify the different techniques, three main families of approaches have been defined, classified as proactive, revision and progressive [Bidot et al., 2008].

3.1.1 Proactive techniques

Proactive techniques are defined by their ability to generate the entire schedule before the execution starts. To perform proactive techniques, knowledge about the stochastic environment and uncertainties is required to produce schedules that are robust or stable or both. Robustness reflects on the quality of the eventually executed schedule compared to the quality of the predicted schedule. The closer the match of the two schedules, the better the robustness, the more reliable the prediction becomes. Stability on the other hand, reflects on the decisions made in the

executed schedule compared to the predicted schedule. If decisions are often changed during execution, the stability will be low. When applying proactive techniques, the focus of the approach is within the generation module, building a global solution at generation time. This means that no reconsideration will be done during execution time.

Proactive techniques consist of three different approaches. The first one is generating a complete generic schedule, which executes correctly in most of the possible scenarios. The second approach is to generate a flexible schedule, where some decisions are postponed until execution time. A flexible schedule is an incomplete schedule where some decisions still have to be made due to constraints restricting the set of complete schedules that can be derived from it. The third approach generates a conditional schedule in which multiple mutually exclusive decisions are developed. The most effective decision will be chosen based on fulfilling some conditions during executing time. A conditional schedule is able to distinct alternative subsets of partially ordered activities that can be modeled. The schedule contains a condition to test for each alternative choice the best outcome at execution time.

With proactive techniques, the schedule should become insensitive to online perturbations by generating a complete, predictive and robust schedule offline and take the worst-case scenario into account. Offline scheduling refers to actions performed before the execution, while online scheduling refers to actions performed during the execution. With proactive techniques a cautious and complete schedule is generated, which desires to maximize the executability by taking into account the possible deviations by using known distributions and compute the solution that has the highest probability of success. Executability reflects on the resilience of the schedule when unexpected events occur. The better the executability, the more a schedule can withstand unexpected events, preventing the need to generate a new schedule.

For generating complete schedules or flexible scheduling, proactive techniques do not require a lot of online memory. Except for the conditional approach, which may require a lot of memory. In general, the computational need is low, since there is no need to search online for solution. Proactive techniques are expected to produce highly robust or stable schedules when there is sufficient knowledge about the environment. However, with the generic approaches and sometimes with flexible schedules, the solution found is not assured to getting the highest quality solution in the observed situation.

3.1.2 Revision techniques

Revision techniques are defined by their ability to react on the schedule by changing decisions during execution when it is necessary or desirable. Revision can occur when the current schedule becomes inconsistent or quality deviates too much. Revision can also be applied when positive events occur like earlier finishes. The revision technique reacts to the current situation and is therefore in need of an execution-monitoring system. This procedure is called rescheduling and is usually called a predictive-reactive approach.

In predictive-reactive scheduling, first of all, a complete operational schedule is generated. Then based on the disruptions of real-time events, the schedule is updated in order to minimize the impact of the disruptions on the system performance. In reactive scheduling, only smaller, partial schedules are generated. When the system requires small adjustments, based on current system status and local information at the machine, new partial schedules are generated. This approach is commonly used in manufacturing industries. In both reactive and predictive-reactive scheduling, the scheduling heuristics determine how frequent revision is required. For reactive scheduling, the size of the set of resources taken into account is less important since partial improvements are suggested. With predictive-reactive scheduling, the impact of the size is larger since a complete new schedule is generated when the scheduling heuristics are triggered.

Revision techniques do not consume a lot of memory online, since only one schedule is stored. On the contrary, the CPU required is quite high, depending on the time spent on rescheduling. Since there is usually not enough time to process a complete search for the optimal solution, the quality of revised solution may degrade, which means that the robustness of schedules is not guaranteed.

3.1.3 Progressive techniques

The idea behind progressive techniques is to interleave scheduling and execution, by solving the problem piece by piece. Each piece corresponds to an activity within the time window. One way to perform progressive techniques is to use a gliding time horizon. This allows the selection and scheduling of new subsets of activities to extend the current executing schedule when more is known about the uncertainties than at the start of the scheduling procedure. In contrary to a flexible proactive approach, where schedules are generated upfront, progressive techniques are able to generate new schedule parts during the execution phase.

There are three time horizons which influence the scheduling of all approaches. First of all, the anticipation time horizon represents the interval between the current time and the expected end time of the last scheduled activity. If the anticipation time horizon becomes too small, one must take immediate action. The second one is the commitment time horizon, which is the temporal window in which decisions are committed which should be final and not be reconsidered. The last one is the reasoning time horizon, which is the temporal window in which possible actions and events are considered in order to make relevant decisions. For progressive techniques to work, it must be clear how the time horizons are defined. Each time window determines when the scheduling should occur and how much time the scheduler has to find an optimal solution.

Progressive techniques use limited online memory, since only parts of the schedule are temporally stored and more is generated along the way. The CPU required is moderate, since the search only solves sub-problems. Decisions made with shorter commitment time horizons, limit the robustness of the schedules, since the global view is lost when the time horizon is glided. At the same time, the stability increases since most of the committed actions will not be altered again.

All three families of techniques are compared to several criteria within Figure 3.1.

	Online memory need	Online CPU need	Quality and robustness	Stability	Commitment time horizon	Knowledge about uncertainty
Revision	Average	High	Low	Low	Very short	No
Proactive generic	Low	Very low	Average	Very high	Long	Yes
Proactive incomplete	Low	Low	High	Very high	Long	Yes
Proactive conditional	Very high	Low	Very high	Very high	Short	Yes
Progressive	Very low	Average	Average	Very high	Short	No

Figure 3.1: The properties of each family of solution-generation techniques [Bidot et al., 2008]

Scheduling in a stochastic and dynamic environment, like the semiconductor industry, is significantly influenced by unexpected events and unpredictable performances. Therefore, maintaining a feasible schedule alone can sometimes be the only goal of scheduling in practice. For the purpose of this research, a combination of proactive and revision techniques is desired, since the algorithm can provide robust schedules and rescheduling can anticipate on the uncertainties. If the HGA is compared to the proactive approaches, proactive generic comes closest since the algorithm creates a schedule prior to the execution phase without an alternative schedule to react to interference.

3.2 Rescheduling

In the previous section, multiple scheduling approaches have been discussed and the way of working will be a combination of generic proactive and revision techniques. The Batch Scheduler HGA

can provide a robust and preferably stable schedule in the generation phase, allowing the shop floor to have proper guidelines on how to handle the current scheduling problem. During the execution phase, problems might occur and rescheduling is required to prevent degradation of the performances [Vieira et al., 2000]. Two questions arise, which will be elaborated. The first one is *how to reschedule* and the second question is *when to reschedule* [Wang and Jiang, 2015].

3.2.1 How to reschedule

How to reschedule focuses on taking the correct resources into consideration for rescheduling. In other words, it tries to repair the schedule in a certain way. There are three common methods to repair a schedule: right shift rescheduling, partial rescheduling and regeneration [Vieira et al., 2003].

Right shift rescheduling postpones each remaining operation by the amount of time needed to make the schedule feasible. It literally shifts the operations to the right side on a Gantt chart. Partial rescheduling reschedules only the operations or resources affected directly by the disruption or poor performance. This method preserves the initial schedule as much as possible, tending to maintain schedule stability. Regeneration reschedules the entire set of operations or resources which are not completed before the rescheduling point, including those not affected by the disruptions. This method is also known as fully rescheduling or complete rescheduling, since no distinction is made between the resources.

3.2.2 When to reschedule

When to reschedule determines the trigger that activates the rescheduling procedure and therefore, the time between two consecutive scheduling points. There are three policies considered: fixed sequencing, periodic review and continuous review [Kizilisik, 2001].

In the fixed sequencing approach, no revision is used. The schedule is only regenerated at the end of its time frame, therefore creating a new schedule when the current one approaches the end. Periodic review monitors the system periodically and determines whether rescheduling is required. The periodic review can be implemented in two ways. First, a fixed time interval, where review periods are equally spaced points in time. Secondly, a variable time interval, where the time between two consecutive scheduling points varies depending on certain performances. The variable time interval makes use of conditional behaviour in order to determine if and when rescheduling is required, making it more responsive to the system. In continuous review, there is constant feedback from the system determining whether the reschedule conditions are triggered as a consequence of changes in the system. In the literature, this policy is also called event-driven scheduling policy.

There is also the possibility to combine a periodic and event-driven approach, which is called a hybrid policy. A hybrid approach validates the system on a set time interval as well as after certain events. These events can range from new jobs arriving to jobs being finished, machine breakdowns or maintenance operations, all triggering the validation of the hybrid policy [Vieira et al., 2003].

3.3 Scheduling in IC manufacturing

Within the IC assembly factories, jobs are assigned by processing priorities and clustered by their product types, which commonly have to be processed on groups of identical parallel machines. Due to the dependency on product type, machine setups and job sequences, the IC scheduling problem is more difficult to solve than the classical parallel machine scheduling problem [Pearn et al., 2007].

Within the IC manufacturing industry, the die bonders are considered the most critical resources [Liu et al., 1997]. It is therefore important to develop efficient scheduling methods to minimize the total die bonder workload and make span and enhance the utilization if possible. The die bonders process different types of dies, each requiring a specific sized chop table, mount head and mount stage. The chop table size is determined by the wafer size, whereas the mount head and mount stage are determined by the die size. When two consecutive jobs have for example

two different wafer sizes, the second job has to be put on hold until the chop table is changed to a new size. The same holds for different mount heads and stages, depending on the die sizes. Thus, the setup time is largely influenced by switching from one product type to another, depending on the size of wafer and die.

When comparing the scheduling approaches to the current way of scheduling within ATSN, a combination of proactive and progressive techniques are applied. The scheduling HGA uses a time window of seven days for each schedule and tries to create a robust and stable solution which minimizes the total die bonder workload. It uses no revision nor conditions to evaluate the schedule which seems to result in a proactive generic approach. However, the factory makes use of a gliding commitment time horizon, determining when choices are fixed and should not be reconsidered. But since no alternative schedules are created in advance to react to the time horizon, a generic proactive approach seems to be used.

Chapter 4

Data Acquisition

In the previous chapters, an introduction and problem definition on the subject are provided, as well as a literature study on scheduling approaches. In the problem definition, it is described how the data can help to provide insights and knowledge on the behaviour of the factory, which can be used as feedback for the system. This data has to be acquired and cleaned first, before it can be analysed. This chapter will look into the process of getting the data from the machines to servers and from the servers to a database.

4.1 From Machine to Server

As mentioned before, all machines are connected to AWACS, allowing the user to download Equipment State Monitoring (ESM) files of each machine. AWACS updates all the data real-time and logs all changes. In order to understand how this data is logged, the ESM file will be elaborated.

ESM files

ESM files contain all the machine specific signal information regarding their production performance. In Figure 4.1, a visual representation of an ESM file is provided, extracted from a die bonder machine. The data within an ESM file is distinguished as standard and non-standard data.

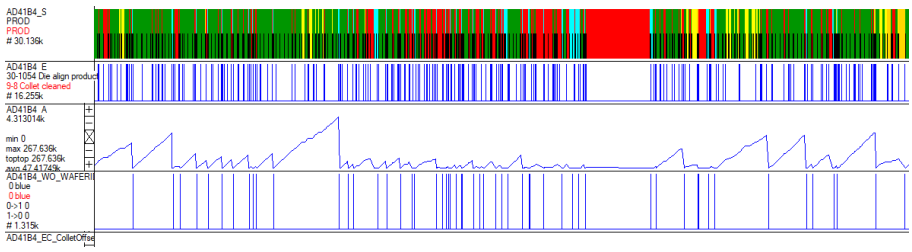


Figure 4.1: Visual representation of an ESM file

Standard data is independent of the type of workstation and contains information on the state, the error and the number of dies processed, all associated with timestamps. The timestamps allow the system to sort the data based on a chronological order, making it possible to track back the behaviour based on real-time occurrences. The state data consists of three different states: up, waiting or down. In up-state the machine is producing, in Figure 4.1 this is indicated by the green color. In waiting-state, the machine could be on hold because of an up- or downstream delay or the machine could be idle because no assignment is provided. At last, when the machine is in down-state, multiple events can occur. The most common down states are caused by an error

state, halted state, setup state or power down. Within the figure, this is mainly indicated by the red or cyan state colors.

The error signal provides a code whenever an error occurs, which is associated with an error message. The amount shows the number of actions processed on a machine. For the die bonder, the amount tracks the number of dies picked from a wafer. This amount can be transformed into the number of products, which will be elaborated later. It can be seen in the figure that the number of dies consecutively picked varies a lot, depending on the inserted wafer.

Non-standard data is dependent on the type of workstation. For the die bonder, the non-standard data logs every wafer id that is inserted, shown in the bottom signal of Figure 4.1. Each die bonder has a wafer table where the wafer is placed in order to be processed. Each wafer has a unique wafer id, which is assigned to a specific order. By using the wafer id, it becomes possible to distinguish different orders in the data. While this is possible to do with the data of die bonder workstations, the wire bonder and multi-plunger workstations provide no data to identify orders, making it impossible to use the data correctly. Therefore, from this point forward, only the die bonder workstations will be analysed. Since the die bonder is always in front of the assembly line and is denoted as the most critical resource, it is assumed the other workstations follow the behaviour of the die bonders [Pearn et al., 2007].

Although Figure 4.1 shows a visual representation of the ESM file, the data itself is organized as a logbook. Table 4.1 shows an example of the logged data. It can be seen that standard data and non-standard data are not logged together but occur as two separate events. Furthermore, the amount and state information are represented in numerical form. However, both have a unique number, representing a false state. For the amount, the false state is 4294967295 ($2^{32} - 1$), which is the highest achievable accuracy since the system uses a binary-counter. For the state it is 255. The false states are either noise in the data or used as a dummy state when no value is correct.

Table 4.1: ESM Log

Timestamp	Amount	Error Code	State Code	Wafer Id
2020-11-03T07:23:14	23440	0	4	
2020-11-03T07:23:36	23449	134447114	5	
2020-11-03T07:23:38	4294967295	0	1	
2020-11-03T07:24:06				W9T05W23-G5
2020-11-03T07:24:07	23456	0	4	
2020-11-03T07:24:08	23469	0	4	
2020-11-03T07:24:47	23484	134447394	5	
2020-11-03T07:25:03	23492	0	255	

While the ESM files provide sufficient information to analyse the workstations, some flaws should not be misinterpreted. The first flaw is the false states, as mentioned above. Secondly, some states on the workstations are automatically logged, but some have to be changed with human interference. The production state and power down are automatically adapted, but others have to be manually inserted. Since this is not done correctly, specific state information can not always be trusted. One of these states is the setup state, which should provide details on the change-over duration. However, it happens often that the change-over is performed while the workstation logs itself in a down state, making it impossible to use the duration of the logged setup states. Finally, the connection with AWACS is not always stable, causing machines to log the same state for a long period of time. According to experts, a workstation is never in the same state for a period longer than 48 hours. If no state change occurs within 24 hours, the machine automatically has to redefine its current state. Therefore, a state period as shown in Figure 4.1, where the machine is in down state (red color) for more than 48 consecutive hours is misrepresented data and should be filtered or cleaned. This will be discussed in Chapter 5.

4.2 From Server to Database

With the data of all workstations available, insights are gained on the production process within the factory. However, the data is insufficient since no knowledge on the order specifications or the product recipe is gained. Order specifications elaborate on the production details such as the order id, production quantity, device type, allocated wafers and the allocated assembly line. The product recipe contains the product details like the die size and position, wire size and material, consume factor and which package and sub-package are associated. Furthermore, it contains information on the required workstation settings, which is essential for the setup time. This information is available through the Manufacturing Execution System (MES).

MES monitors and synchronizes manufacturing activities across globally distributed plants, sites and vendors. It provides real-time information on product and order details of the plant floors and combines it with the financial and planning systems to provide all information.

By creating a connection with the MES and using the unique order and product identifiers, it is possible to extract the required order specifications and product recipes.

4.3 Tracking Database

To analyse the data, all information should be gathered in one place for a clear overview. Therefore, a tracking software is developed, which connects to all data sources and combines the information into a tracking database. The tracking software starts out with downloading and updating all ESM files from the AWACS to a local storage point. Based on a machine list, all ESM files are checked. It compares the last timestamp of the local and online file and updates the local file when necessary. Next, the database is initialized and verifies whether all machines are taken into account. Once the database is set, it starts updating all machines individually.

First, the associated ESM file is extracted and a non-standard variant is created. In Figure 4.1, the last signal represents all the wafer ids, each line indicating a wafer scan. The non-standard variant tracks the timestamp of each wafer scan and by using the MES system information, the wafer id is transformed into an order id. The resulting non-standard variant provides the start and end time of each order. The tracking software jumps from one order id to the next, recording all associated data. It might happen that the wafer id cannot be transformed into an order id, due to the lack of information. To prevent two orders from merging unintentionally, unassigned wafers are assigned to dummy orders, which can easily be recognized within the database. Figure 4.2 provides a visual representation of the combined ESM file, where the dashed green line indicates unique work order ids. All signals between the dashed green lines are wafer scans, assigned to the order.

Secondly, after all the orders of each machine are listed in the database, the metrics can be updated. The first metric taken into consideration is the start time. Each start time is caused by the detection of a wafer id within the system. It is therefore assumed that two consecutive wafer scans happen without interference. Since this is often not the case due to setups, errors or delays, a new variable is introduced: the start time corrected. Based on the time stamp of the wafer scan, the software looks backwards in time in the data until it finds the last timestamp where the amount has been increased. This entry represents the last production event and finishes the previous order. This point is denoted as start time corrected and everything afterwards up until the wafer id is part of the new order. In Figure 4.2, the dashed red line indicates a start time corrected with a clear difference from the wafer scan. The start time corrected is determined for each order of each machine and is recorded in the database. The duration of each order is determined by the difference of the start time corrected of the order and the start time corrected of the consecutive order.

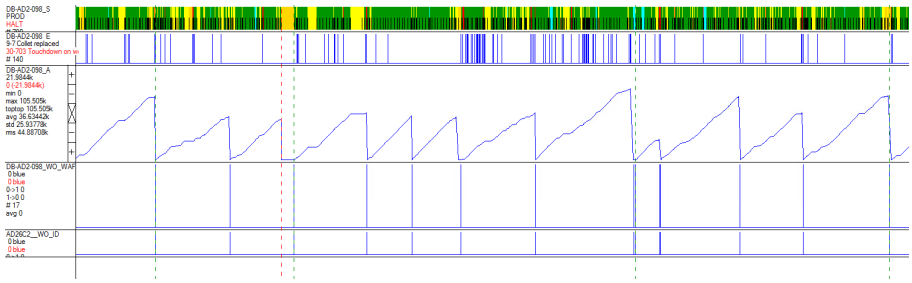


Figure 4.2: Combined ESM file

The second metric is the amount, which represents the number of dies processed within each order. For each wafer, the number of dies picked is recorded as the amount and each new wafer resets the amount back to zero. Since the total number of dies is desired for each order, the sum of the amounts between two order ids is taken as the total number of dies.

Simultaneously with updating the amount, the duration of each state is recorded. The software loops over each entry of standard data and checks if the state changes. It records the first time a state occurs and when the state identifier changes. The duration is determined by taking the difference between the time stamps. For each order, the durations are summed for each possible state. There are 14 different states possible, including the 255 dummy state.

The last metric is the number of wafer scans that have occurred during each order. The total number of wafer scans are recorded, as well as the total number of unique wafer scans. It often happens that a wafer is scanned multiple times due to an error. This could be caused by a failure or error, a power down or an invalid setup.

Finally, after all the metrics are updated, the software starts on the order and product details. Using the order id of each entry in the database, the associated product id can be retrieved. The software creates a connection with the MES, using a SQL server. With the use of queries, information is traded between the database and the MES. By using the order id and the product id, all required information as stated in Section 4.2 can be extracted and logged into the database.

If the database starts out empty, the software initiates all machines and fills the entire database. If the database is already filled but not up to date, the software updates the database. Since it is key to keep the data as accurate and up to date as possible, the tracking software updates the ESM files and the database every hour.

For this research, the used database consists of 71.626 entries over 51 columns. The data is extracted from 17 months time, dating from 26-05-2020 to 11-10-2021, and contains 26.131 unique orders. Furthermore, 91 different assembly lines are recorded with a total of 261 die bonders. The data within the database will be explored in the next chapter.

Chapter 5

Data Cleaning & Exploration

In the previous chapter, data acquisition is elaborated, explaining what data is necessary and how it is acquired. This chapter will focus on the analysis of the data and what steps are required in order to filter incorrect inputs and noise.

All data analysis is performed in python, mainly with use of pandas. All python scripts are coded in Visual Studio Code using version 1.63.0.

5.1 Basic Data Cleaning

When analysing data, it is important that the data itself is consistent and valid. All entries in the database should correctly contribute to the averages and statistics. The database created as explained in Chapter 4.3 has 71.626 entries. However, not all these entries are valid and usable. Entries could have empty or NaN values in any of the 51 columns, which results in errors or invalid analyses. To solve this problem, the database is subjected to a cleaning script. The following steps are taken to clean the data:

- Jobs without processing times are removed
- Jobs without a start time corrected are removed
- Jobs with any negative duration are removed
- Jobs with an amount smaller than 1 are removed
- Jobs with an engineering indicator (maintenance jobs) are removed
- Jobs without a consecutive job are removed
- Jobs without a previous job are removed

Jobs without a consecutive job are still in process while the database was constructed. Jobs without a previous job are starting jobs and are neglected due to a lack of prior knowledge. After this basic cleaning is applied, a more in-depth validation of the quantities can be performed.

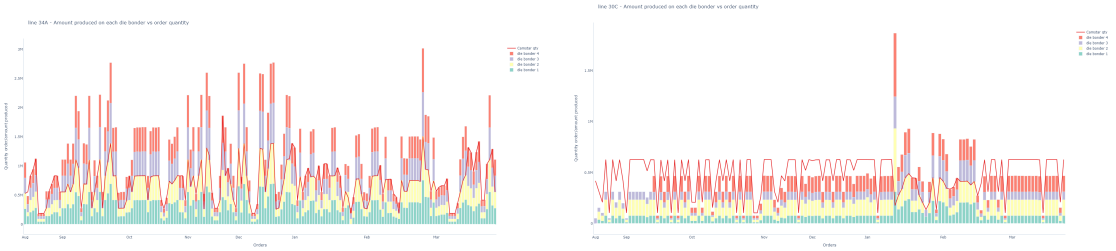
5.2 The transformation of quantities

One of the major parameters to analyse the performance of a factory is the throughput. The throughput represents the output over a period of time. For the assembly lines, the throughput is determined by the products produced, which is dependent on the production speed of the assembly lines. Unfortunately, not all acquired data can simply be used due to noise or incorrect transformations. Therefore, all important quantities will be analysed and corrections will be performed where necessary.

Consume Factor Corrected

The first quantity is the amount. Amount can be misinterpreted by different levels within a company. For management, amount could represent the number of orders completed. For the factory, amount represents the number of products produced, whereas for the die bonders, amount represents the number of dies processed. The latter one is the value taken into consideration within the tracking database. To transform the number of dies to the number of ICs, the consume factor is required. The consume factor represents the number of dies that have to be processed in order to fulfill the product requirement. Within the MES, most products have a consume factor. Therefore, it would be expected that the number of products can simply be determined by acquiring all consume factors and divide the number of picked dies by the consume factor.

However, after applying this method and analysing the data, several orders seemed off. The problem is visualized in Figure 5.1. Each bar represents an order processed on a specific assembly line. Each color within the bar represents a specific die bonder and the height represents the amount produced. The red line indicates the order quantity, which should represent the total ICs produced on the assembly line. However, it can clearly be seen in Figure 5.1a that the consume factor is too low, causing the output to be double the required quantity. In Figure 5.1b the consume factor is too high, causing the output to be lower than required.



(a) Overproduction due to off Consume Factor

(b) Underproduction due to off Consume Factor

Figure 5.1: Incorrect consume factor

To solve this, each order is validated. When an order is assigned to an assembly line, the production quantity is spread among the different machines. Since each machine is separately tracked, an order is commonly distributed over multiple observations. All these observations are indicated with the same unique order id, making it possible to find all observations that contributed to one order. By taking the total produced amount, adjusted with the consume factor, and dividing it by the order quantity, a production ratio is determined. If the consume factor is correct, the ratio has a value of 1. A ratio higher or lower represents an over- or underproduction.

Since consume factors are consistent within subpackages, all observations from each subpackage are reviewed on the production ratio. If the average production ratio is off, the consume factor is corrected in order to adjust the production ratio to 1. It is important to notice that the consume factor can only have an integer value between one and four. For every subpackage, the orders are reviewed with the corrected consume factor to qualify the improvement. Based on this procedure, 18 out of the 65 subpackages received an adjusted consume factor called consume factor corrected. Using the new consume factor corrected, the amount can be correctly validated.

Production Speed

The second metric to discuss is the production speed of each machine. The speed of a machine is influenced by several parameters like the device type, the consume factor and the machine model but also the machine down times and setup times. The first three parameters are deterministic variables but the latter two are stochastic times, mainly influenced by unexpected events on the shop floor. Therefore, two indicators of speed are introduced: the production speed and the

effective production speed. The production speed is the speed while the machine is producing, determined by dividing the produced ICs over the duration of the production state. The effective production speed is the total speed of the order, defined as the produced ICs divided by the total production time. By comparing the production speed, no external influences are taken into account.

Die sizes

The last metric to discuss is the die sizes in the database. Every order produces one specific device type, which has a required die size. Even though most of the orders are filled with information during the tracking software update, some values have not been found online and are therefore empty. If the missing information would be related to machine performance, the entry would have to be filtered. If the information is related to product specifications, the information could be subtracted from other entries. In some cases the information is manually implemented with a human error in the loop. Incorrect implementation of a metric causes invalid data. A mix-up between micrometer and centimeter could make the entire entry invalid.

Therefore, a filling heuristic for the die sizes has been written. First, the heuristics loops over similar entries to try and find the missing product information. Secondly, values larger than thousand are transformed back to centimeters by dividing them by a thousand. Thirdly, the die is commonly a square object resulting in the die size x and die size y to be equal. If one of the two attributes is empty, it is filled with the data of the other. Finally, when both values are empty, the average die size of the subpackage is taken.

5.3 Cleaning Outliers

Since all important metrics have been validated and corrected where required, the data can be analysed for critical outliers. Outliers are entries which do not correlate with comparable entries. If the difference is small, the outlier is considered noise but as the difference grows larger, the influence on the total data increases and the entry is considered a critical outlier. For this research and database, two indicators are evaluated to detect outliers. First the production speed is compared, which is mainly dependent on the machine model and the subpackage. Secondly, the production ratio is considered, which indicates the proportion of the total order that has been produced. Since it is desired to keep the database as large and authentic as possible, entries are only filtered when absolutely necessary.

For the production speed, it is determined by ATSN that no machine can produce faster than 48.180 ICs per hour. Therefore, all production speeds above this limit will be invalid. It is expected that production speeds above this limit are caused by noise within the ESM data. The lower boundary has also been investigated, but it appeared that the influence is too small to implement a lower limit.

For the produced amount, the analysis was more complex. Due to the orders not being completed in one event and the possibility that the same order is distributed over multiple lines, the produced amount could become invalid. Furthermore, the factory could under- or overproduce a specific product for reasons untraceable from the available data. Since no conclusions could be drafted on lower values, no lower boundary is implemented. Extremely high ratios are an indication that the amount is incorrectly logged. There are only several entries with a production ratio above ten, which are filtered from the database.

5.4 Line Selection

Cleaning the database, transforming quantities and clearing outliers all resulted into a more accurate and usable database. However, not all data is relevant for the purpose of this research. Since the desired data is associated to the results of the Batch Scheduler, only the assembly lines that are scheduled with the HGA are considered of interest. Currently, a total of 35 assembly

lines are dedicated to the HGA with a total of 109 unique die bonders. Therefore, the database will be reduced to entry logs of these 109 machines. The entries are obtained from 14 months of data. Details on the database are enlisted in Table 5.1.

Table 5.1: Cleaned and filtered database details

Total entries	19.483
Unique orders	7031
Total days	422
Earliest order	2020-08-11
Latest order	2021-10-07
Assembly lines	35
Unique die bonders	109
Device types	473
Packages	6
Subpackages	29

Chapter 6

Simulation

In the previous chapters, data acquiring and exploration resulted into better understanding the entries and filtering unwanted data. The resulting database provides the knowledge to detect patterns and to make educated guesses on the historical behaviour. However, no knowledge is currently gained on the scheduling performance and the influence of rescheduling. Therefore, a simulation model of the factory is required. The model will mimic the real-world environment, providing the possibility to simulate and evaluate the real-world processes and performances. This chapter will first explain what data is required to create the model and how this data is prepared. Secondly, the simulation setup will be elaborated, explaining the CSSL library and the functionality of integrated scripts. Afterwards, the simulation settings like the length and warm up period will be determined. Finally, the rescheduling heuristics will be explained and implemented.

The simulation is constructed in C-Sharp using Microsoft Visual Studio 2019. Furthermore, the basic framework of the simulation is provided with the implementation of the so-called CSSL library [Adan et al., 2020].

6.1 Data preparation

The simulation model requires input of the historical database to mimic the real-world behaviour. Before the data can be used in the model, some preparation is required since the simulation is unable to read python data frames. Instead, all data will be transformed from pandas into Comma Separated Values (CSV) files, which are compatible and relatively easy to implement. CSV files are plain text files, containing lists of data, separated by commas. With five steps, the simulation is supplied with five CSV files, containing all required data from the database.

Machine Information

The first step is to define the machine information. The same assembly lines as mentioned in Chapter 5.4 will be used in the simulation, thus a total of 35 lines. For each assembly line, the number of machines and the machine model are obtained from the database. In addition, an Excel sheet is supplied by Nexperia containing the device types that can be produced per assembly line. This information is required for the scheduler to determine which product can be produced on what lines. An example of the information is shown in Table 6.1.

Table 6.1: Machine information

Assembly line	Number of machines	Machine model	Device types
13A	2	ADAT 3 48K	S_SD, S_DD, S_DD_6W, S_DD_8W, S_TD
31B	4	ADAT 2 18K	S_SD, S_SD_2W, S_DD

Device Information

After the machine information is gathered, the device information database is constructed. A list is created of each device type obtained from the database, taking the attributes into account that influence the setup time. This list is filled with additional product information, obtained from a Nexperia Excel sheet. An example of the information is provided in Table 6.2.

Table 6.2: Device information

Device name	Device number	Device type	Consume factor	Wire size	Wire material	Die size	Wafer size	Subpackage	Order frequency	Production quantity
BAS32PY	93345	S_DD	2	22.5	Cu4N	0.25	6	SOT363V_APM	2	[207937, 209221]
NX128AK	93376	S_SD	1	23	Au4N	0.37	8	SOT363AF_TS	3	[533011, 2859, 1130216]
PBS2515	91409		4	35	Cu4N	0.4		SOT363M1_APM	1	[74377]

As seen in the last row of the table, not all information can be obtained for each device type. This is because not every device type obtained from the historical database is available in the Excel sheet. Due to missing information, a total of 284 out of the 473 devices are removed. The remaining database has 5548 unique orders with a total of almost 15.900 entries.

In addition to the device information, all production quantities that occurred in the historical data are listed as well. The last two columns in Table 6.2 show the order frequency and sizes of the production quantities for each device type. The production quantities will be used to randomly sample a possible order size in the simulation.

Machine Learning data set

While working on this research project, another research has been conducted on the same real-world data set, by J. Vermunt [Vermunt, 2021]. In this research, a machine learning algorithm is developed which aims to improve the accuracy of the predicted Effective Process Times (EPT) of incoming orders. Currently, EPTs of incoming orders are determined using a set of rules determined by factory planners, using a few attributes of the assembly line and the product. The machine learning algorithm uses the historical data as input to derive more accurate predictions, by using more attributes and data. The obtained model successfully improved the prediction accuracy and thus, it was desired to implement the machine learning model in the simulation.

The machine learning model was transformed into an Open Neural Network Exchange (ONNX) model, which is compatible with the simulation. To create the ONNX model, the historical database is required as input but has to be adapted into a specified format. After the historical database has been adapted, the resulting ONNX model can be implemented in the simulation model. It is important that the input for the ONNX model is identical to the input of the simulation model. Therefore, after creating the machine and device information databases, all entries are combined to create the required database for the ONNX model. The resulting output model matches the generated CSV files and can be implemented into the simulation.

Production speed

The effective production speed is one of the stochastic values which are hard to correctly implement in the simulation. In the real-world factory, all sorts of disturbances could influence the effective production speed. In order to mimic this behaviour, disturbances and production speeds are separately implemented.

The production speed will be sampled from the historical data. To enable the sampling approach, the data is split into subpopulations, based on one order attribute and one assembly line attribute. During the research of Vermunt, it was concluded that the machine model of an assembly line is the strongest prediction variable and will thus be used as the assembly line attribute. For the order attribute, the consume factor is chosen. The consume factor is deemed an important predictor by planner expertise and has great influence on the production speed. Combining

these two attributes results in twenty subpopulations. The observation count per pool is shown in Table 6.3.

It can be noted that the observation count is unequally distributed over the subpopulations. The ATSN factory contains an older and new die bonder model, called *ADAT 2* and *ADAT 3*, with two and three different versions respectively, resulting in five machine models. The older models are unable to produce devices with a consume factor of three or four, resulting in empty pools. The *ADAT 3 Glue* is dedicated to produce devices with consume factor three, resulting in empty pools for the other consume factors. From the twenty subpopulations, seven are empty, four contain less than 50 entries and nine contain more than 50 entries.

Table 6.3: Observation count per model and consume factor subpopulation

CF / Model	ADAT 2 14K	ADAT 2 18K	ADAT 3 36K	ADAT 3 48K	ADAT 3 Glue
1	56	5	51	46	0
2	10 071	463	992	3346	0
3	0	0	105	422	37
4	0	0	37	237	0

To enable sampling from a pool, a distribution has to be fitted. Subpopulations with more than 50 observations can be fitted with an empirical distribution from which the production speed can be sampled.

For the subpopulations with less than 50 observations, it is chosen to fit a log-normal distribution. The log-normal is a nonnegative distribution, which is desired since production speeds are nonnegative. But the log-normal is also a right skewed distribution, whereas the production speed is a left skewed distribution. To fit the log-normal distribution, the production speed must be normalized to create a right skewed distribution. The method of Vermunt [Vermunt, 2021] is followed to create this normalized distribution of the production speed.

To fit the log-normal distribution, the mean μ and variance σ of the subpopulation are required. It is assumed that each subpopulation p has a different μ_p and an identical σ, proportional to μ_p. Using the mean of the subpopulation, the variance can be determined using Equation (6.1).

$$\sigma_p = \mu_p \cdot CV_{global} \tag{6.1}$$

CV_{global}, the global Coefficient of Variation of all subpopulations, is determined by the average CV of the subpopulations with more than 50 observations. With the determined μ_p and σ_p, the log-normal variables $\mu_{ln\ p}$ and $\sigma^2_{ln\ p}$ can be determined using Equation (6.2) and Equation (6.3).

$$\mu_{ln\ p} = ln(\ \mu_p) - \frac{1}{2}\sigma^2_{ln\ p} \tag{6.2}$$

$$\sigma^2_{ln\ p} = ln(1 + (\frac{\sigma_p}{\mu_p})^2) \tag{6.3}$$

With these derived variables, the log-normal distribution can be fitted. The mean of the log-normal distribution is determined using Equation (6.4). The variance of the log-normal distribution is determined using Equation (6.5).

$$E[X] = \exp(\mu_{ln\ p} + \frac{\sigma^2_{ln\ p}}{2}) \tag{6.4}$$

$$Var[X] = [\exp(\sigma^2_{ln\ p}) - 1] \cdot \exp(2\mu_{ln\ p} + \sigma^2_{ln\ p}) \tag{6.5}$$

With the distributions fitted to all the subpopulations, the production speed can be sampled based on the machine model of the assembly line and the consume factor of the device type in production.

Down Ratio

Whereas the production speed represents the performance while the machine is in production state, the effective production speed represents the overall machine performance. The effective production speed is greatly influenced by the down time, which represents the time spent in any other state than production or setup. The performance in production state is only determined by the machine itself. As a result, the production speed depends on the product and machine attributes of each individual machine. The down time is determined by all machines in an assembly line, since one error generally causes the entire line to hold or fail. As a result, the down time only depends on the product attributes and is applied to the entire line at once.

To apply the down time in the simulation, a distribution has to be fitted. Preferably, one overall distribution can be applied to all observations. The first approach to analyse the down time distribution was to execute a histogram plot of all observed down times. The observations are distributed in subpopulations, based on the consume factor. The resulting histogram plot does not present a clear overview, due to some extremely large values. By taking a subset of the data with all values above 100 hours down time removed, 99.55% of the data is still used while the plot becomes presentable. The result is shown in Figure 6.1a.

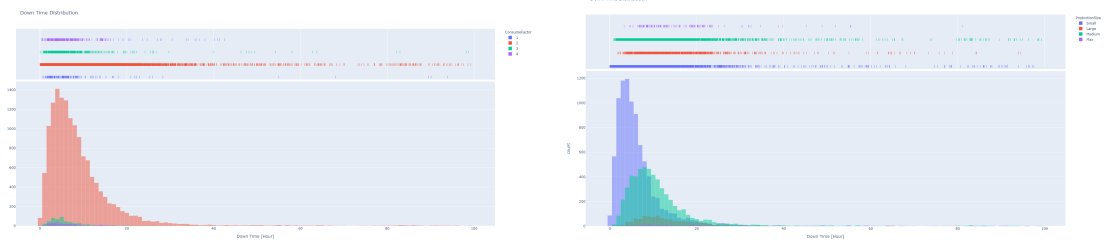
The histogram uses 100 bins which are equally distributed between 0 and 100 hours. In the top of the figure, tally marks are shown for each observation, giving an indication how it is spread. Due to the unequal observation count of each consume factor, no overall distribution could be defined. Neither do the four subpopulations present a clear distribution.

While analysing the data, it was assumed that the down time would be strongly dependent on the order quantity. The larger an order, the longer it takes to process and the more chances occur for disturbances to arise, causing down times. Therefore, the observations are distributed into four new subpopulations, based on the production quantity. The largest quantity is determined, with an order quantity of 1.86 million products, and the four groups represents a quarter of this quantity. The new distributions are shown in Table 6.4.

Table 6.4: Production size groups

	Small	Medium	Large	Max
Production Size [%]	0 - 25	26 - 50	51 - 75	76-100
Observations	9072	5554	1109	133

The new subpopulations are more balanced and the resulting histogram is shown in Figure 6.1b. Again, the 72 entries above 100 hours are removed for visual purpose. The figure shows a relation between larger orders and higher down times, but no proper distribution can be fitted on the data due to the large variation.



(a) Down time distribution based on consume factor (b) Down time distribution based on order quantity

Figure 6.1: Down time analysis

Instead of taking the absolute down time, it is also possible to approach it as a relative value. This value is called the down ratio and represents the ratio of the total time of an order that is not

spent in production or setup state. The down ratio is the calculated as shown in Equation (6.6).

$$\begin{aligned} R_D &= \frac{T_T - (T_P + T_S)}{T_T} \\ &= \frac{T_D}{T_T} \end{aligned} \tag{6.6}$$

R_D represents the down ratio, whereas T_T is the total time it took to finish the order and T_D the time spent in down state. T_P stands for the total time spent in production state and T_S is the time spent in setup state. All times are expressed in hours. Using the down ratio, another histogram plot is created with the observations categorized on the production quantity. The resulting plot is shown in Figure 6.2.

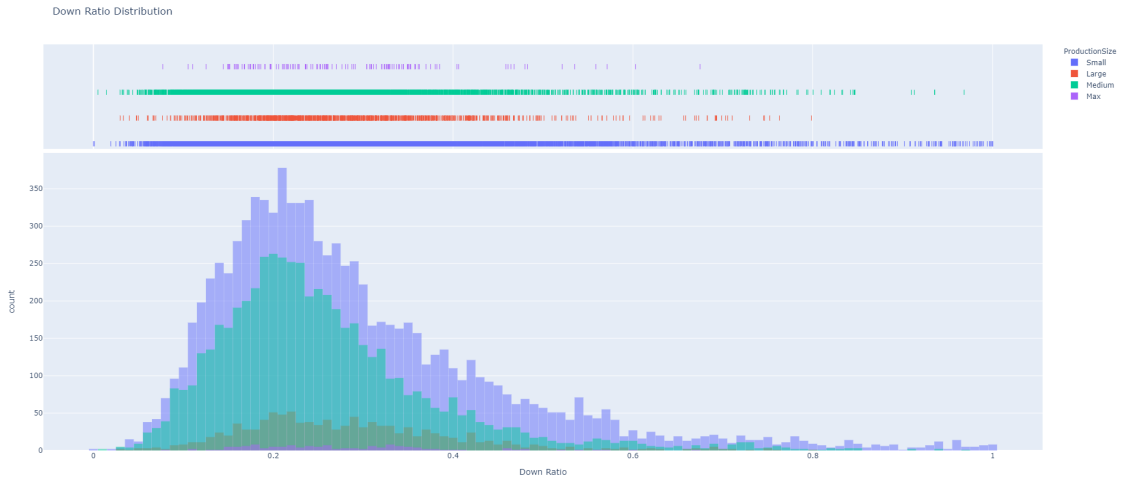


Figure 6.2: Down ratio distribution based on order quantity

The down ratio figure shows all four subpopulations following a similar distribution, with all the medians centered around 0.23 and the overall average at 0.278. Because all four subpopulations follow the same distribution, one overall distribution can be fitted to all observations. As a result, the down ratio can be implemented without being dependent on the consume factor or order quantity.

Nevertheless, Figure 6.2 shows some of the smaller orders having extremely high down ratios. This is caused by observations where only a few products are produced but a really large down time is registered in comparison to the total completion time. These extremely high down ratios are unrealistic and should be seen as outliers. If these outliers would not be filtered, larger orders could be assigned with extremely high down ratios, causing extremely high down times. These high down times are never recorded in the factory and should not be present in the simulation. To solve this problem, the highest down ratio present in the Large order subpopulation is taken as the upper limit for all orders, which is equal to 0.80. Since all subpopulations have observations close to zero, no minimum boundary is required.

For each observation in the database, the down ratio is determined and listed. All values above the upper limit are removed. After the completion of the down ratio database, all five databases are completed and transformed into CSV files. The use of the data within CSV files will be elaborated within Section 6.2

6.2 Simulation Setup

As previously mentioned, the simulation is created in C-sharp and uses the CSSL library framework as the foundation[Adan et al., 2020]. First of all, the framework itself will be briefly elaborated. Secondly, the model is developed, aimed to mimic the ATSN performances. Afterwards, the simulation settings will be explained. Finally, the rescheduling heuristics will be explained.

6.2.1 CSSL Framework

The CSSL library is a class library for discrete event simulation (DES). DES is a highly efficient simulation technique to replicate the behaviour and performances of systems. The CSSL library contains seven components, which are illustrated in Figure 6.3. The settings and utilities are predetermined and can be altered in the program settings. The simulation component is the main component and contains the other components, which define for example the executable actions and events, the length and number of replications, the observers and most importantly, the model itself.

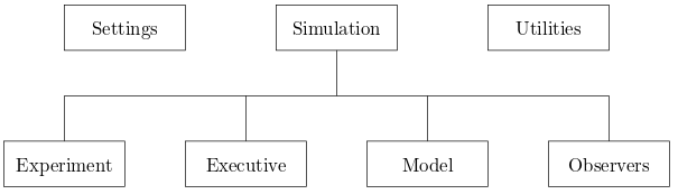


Figure 6.3: CSSL library components

When using the CSSL library, the main goal for the user is to implement a correct and complete model. The other components mainly allow the user to easily alter settings and execute the model. Therefore, all information provided next will be aimed to create the model.

6.2.2 ATSN Model

Before the model can be developed, the framework first has to be validated. Since it is assumed that the model requires single-server $M/M/1$ systems and multi-server $M/M/s$ systems, both queuing networks are validated. The validation is elaborated in Appendix B and it is concluded that both systems are correctly implemented.

With the validation correctly performed, the model can be expanded to replicate the ATSN facility. The current way of working with the Batch Scheduler is focused on weekly schedules. Therefore, the simulation will use a weekly job arrival rate and a weekly schedule generation rate. The accuracy of the input in simulation is in seconds.

First the overall factory component will be elaborated. Afterwards, the capacity generator will be explained in more details, followed by the scheduler. At last, the performance trackers and indicators will be discussed.

Factory

The simulation model is divided into several components, each dedicated to performing a specific task. All these components are connected to the overall factory, which can be seen as the parent element. The simulation starts at the capacity generator, which mimics the real-world supply chain department. Jobs are generated on a weekly interval, based on the performance and capacity of the system. The jobs are distributed to the scheduler, which assigns the jobs to the available resources, creating multiple possible schedules. The best schedule is determined and implemented on the shop floor. The shop floor exists of 35 assembly lines, each containing a specified number of machines. The assembly line determines the duration of an order, and after completion the order

is removed from the model. This process is repeated until the end of the simulation. A simplified schematic overview of the factory is presented in Figure 6.4.

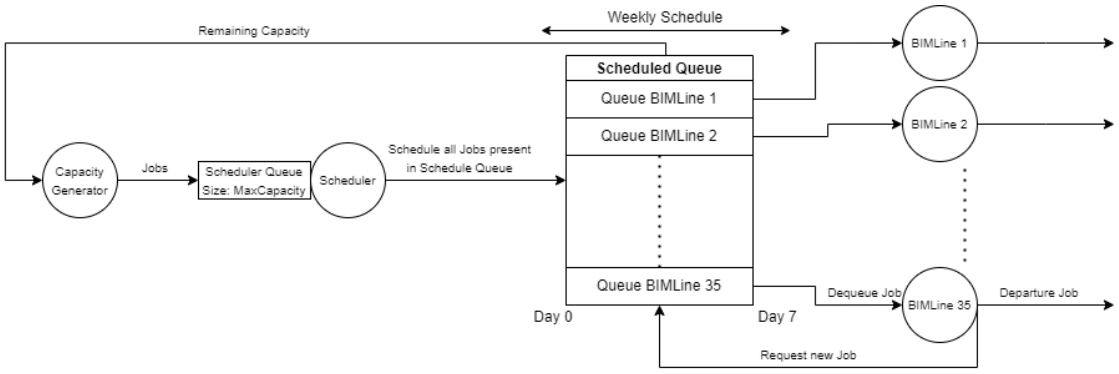


Figure 6.4: Simplified schematic overview of the factory

As seen in the figure, each assembly line is equipped with a queue for jobs, which is filled by the scheduler. Whenever the BIM line is idle and receives jobs in the queue by the scheduler, the production process is initiated. The job is retrieved and the setup time is determined. The setup time is based on the current assembly line settings and the required device type settings. This approach is deterministic, where each job requires the basic setup, and for each element that requires a change-over, more time is added. All possible change-overs and the associated required time are presented in Table 6.5, where the setup time is the sum of all change-over times. If the job requires for example a change in the die size and the wafer size, the total setup time will be $T_S = 32 + 60 + 60 = 152$ minutes.

Table 6.5: Assembly line change-over times

Setting	Setup Time [min]
Basic Setup	32
Die Size	60
Device Type	60
Wafer Size	60
Wire Material	360
Wire Size	60

After the setup time is determined, the production time is set. The production time is calculated by dividing the production quantity by the production speed of the assembly line. The production speed is determined with a stochastic approach.

At the start of the simulation, each assembly line creates a list containing four speed distributions. These distributions are fitted based on the machine model and possible consume factors, as explained in Section 6.1. Based on the machine model of the assembly line and the consume factor of the job, a distribution is selected. With this distribution, a random production speed observation from the specified subpopulation is retrieved for each die bonder in the assembly line. The sum of the sampled production speeds represents the production speed of the assembly line.

After the production time is determined, the down time is calculated. Each assembly line contains a down time distribution containing all 15.900 observed down ratios, varying between 0.00 and 0.80. One of these down ratios is randomly sampled, and by rewriting Equation (6.6) to Equation (6.7), the down time is obtained.

$$\begin{aligned} T_D &= R_D \cdot T_T \\ &= \frac{T_P + T_S}{1 - R_D} - (T_P + T_S) \end{aligned} \tag{6.7}$$

$$T_T = T_S + T_P + T_D \tag{6.8}$$

T_D represents the down time, T_P is the production time, T_S is the setup time and R_D is the sampled down ratio. With the setup time, production time and down time determined, the total duration of the job is obtained as shown in Equation (6.8), with T_T as the total time to complete a job. This duration is used to schedule the departure event of the job. After the job is completed and removed, the assembly line sends a request to the queue for a new job. If this request is fulfilled, the same procedure is repeated.

Capacity Generator

At the start of each week, the capacity generator determines how much capacity can be distributed to the factory and generates jobs in order to meet this requirement. It mimics the behaviour of a real-world supply chain system, providing a weekly workload based on the current performance and remaining workload. At the start of each week, the capacity generator retrieves all jobs in the system that are still in the queue of an assembly line. The simulation is non-preemptive, thus all jobs in process will not be disrupted since the production process of a job is seen as one event.

After retrieving all jobs, the predicted workload of these jobs is calculated. Within the factory, the capacity of a job represents the predicted workload for an assembly line. Since the jobs are not yet assigned to an assembly line in this state, the predicted workload cannot be determined and an alternative workload is required. Each job contains an expected duration, which represents the expected average duration based on the device type. The expected duration is calculated using Equation (6.9).

$$T_{ed} = \frac{\frac{Q}{E[V]} + T_{bs}}{1 - E[R_D]} \tag{6.9}$$

T_{ed} represents the expected duration, with Q, the production quantity, and $E[V]$, the average production speed of the device type. $E[R_D]$ is the mean down ratio of all observations and T_{bs} is the basic setup, which is applied to all jobs. By using these parameters, all attributes that can affect the job duration are taken into account.

After the total expected workload of the retrieved jobs is determined, using the expected durations, the *weekly maximum capacity* can be calculated. The *weekly maximum capacity* determines how much workload can be distributed to the system, and is adapted based on the performances and remaining expected workload. First, the *weekly maximum capacity* is equal to the interval time of the generator and the number of machines present in the simulation. With 109 machines and a weekly interval time (168 hours), the maximum capacity is equal to 18 312 hours. This value has to be reduced with the remaining workload, determined by the retrieved jobs. There is also an incremental value, called the idle time. Every week, the total idle time of the machines is determined and is added to the maximum capacity. This is done to prevent machines from being idle by providing more workload. The *weekly maximum capacity* is determined using Equation (6.10).

$$C_m = \sum_{i=0}^{L} (M_i \cdot T_w) - C_R + T_I \tag{6.10}$$

C_m represents the *weekly maximum capacity* in seconds, with L being the number of lines (35) and M_i the number of machines in line i. T_w represents the number of seconds in one week. C_R is the remaining workload, determined by the retrieved jobs. T_I is the total weekly idle time over the previous week.

After determining how much capacity can be distributed, jobs have to be generated. A new variable called the *generation capacity* is introduced, and is set equal to the *weekly maximum capacity*. While the *generation capacity* is larger than zero, jobs will be generated.

When generating a job, the device type has to be set first. For each job, a random number is sampled between zero and the total order count and this number determines which device type is generated. The higher the occurrence frequency of a device type in the historical data, the larger the probability that it will be generated.

Secondly, the production quantity has to be set. All historical production quantities are stored per device type and one of these quantities is randomly sampled and set for the job. Using the device type and production quantity, the expected duration is calculated for using Equation (6.9).

After the job is generated and set, the job itself is assigned to the scheduler queue and the expected duration is subtracted from the *generation capacity*. This process is repeated until the *generation capacity* exceeds zero. After the generation has stopped, the next generation event is scheduled with a week interval.

A schematic overview of the process is provided in Figure 6.5. The max capacity is determined by the interval period and the number of machines. The maximum capacity is adapted by the workload of the retrieved jobs and the total idle time of the previous week. Based on the maximum capacity, the generation capacity is set. Each generated job reduces the generation capacity with its expected duration and the job is assigned to the scheduler queue.

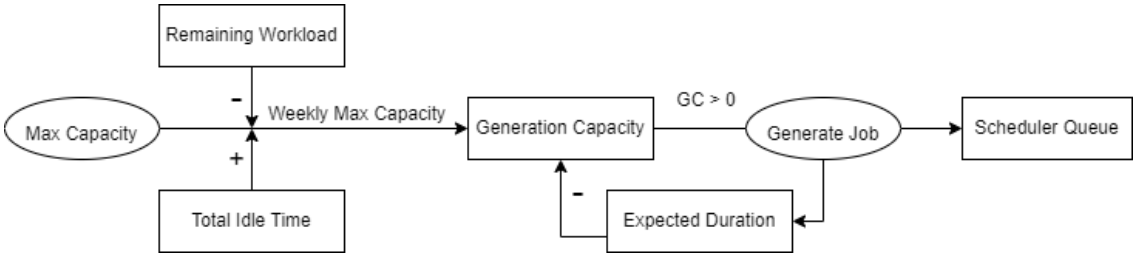


Figure 6.5: Simplified schematic overview of the capacity generator

Scheduler

After the generated jobs and the retrieved jobs are assigned to the scheduler queue, the scheduling procedure can start. The scheduler consists of three parts: the schedule generator, the data writer and the ONNX EPT predictor. The schedule generator is the overall component that uses the other parts to derive a schedule. The schedule generator, also called the scheduler, generates schedules using the Batch Scheduler hybrid genetic algorithm, developed in previous research [Adan et al., 2018]. In a generic algorithm, a population of possible solution within the boundaries of the search space evolves toward better solutions through an iterative evolutionary process. During each generation, the fitness of the solutions are evaluated against the objective function. The best solutions are defined, and by applying a cross-over mechanism and a mutation operator, a new generation of solutions is derived. This procedure is repeated over and over to find the best objective score. The objection function used in this research tries to minimize the maximum completion time of each job, also known as minimizing the make span or as the C_{max} objective.

The scheduler follows several steps to derive schedule. Before the scheduler can start, all jobs have to be obtained. If all generated jobs and retrieved jobs are assigned to schedule queue, the scheduler continues with data preparation. The HGA requires a specific formatted data file as input. To create this data file, the data writer is used.

The data writer follows six steps to create the correct data file. First of all, all assembly line information and job information are transformed into new classes. Then, for each combination of job and assembly line, it is determined if they are compatible with respect to the producibility, using the data from Section 6.1. Next, all production times are predicted using the ONNX

EPT predictor. Each combination of job and assembly line is checked to determine all possible production times. After the production times are predicted, the setup times for each combination of two sequential jobs are determined. Aside of the setup time between two jobs, the setup time required for each combination of assembly line and job is determined as well. All this information is transformed into a data file which can be read by the HGA.

The ONNX EPT Predictor is developed with the research of J. Vermunt [Vermunt, 2021], making better predictions for the Effective Production Time (EPT). The algorithm has been transformed into an ONNX model, which has been implemented in the simulation as the ONNX EPT Predictor. Furthermore, the model is adapted to output production times instead of effective production times, since the down time is not taken into account within the scheduling process. Before the ONNX model, the scheduler used a simple deterministic heuristic, where a combination of the product device type, the production quantity, the assigned assembly line and the wafer size resulted in an expected process time. However, that approach uses outdated production speeds and provides less accurate predictions than the ONNX model. The ONNX model uses a more complex heuristics, with more product specifications and is completely updated and synchronized with the data of the simulation.

After the HGA is supplied with the input data file, the algorithm is executed for a specified number of runs and evolutions, each run trying to minimize the objective function. The output file of each run contains the objective score and the job allocation sequence. The best objective score is determined and the jobs are allocated to their specified assembly line.

A schematic overview of the scheduler process is shown in Figure 6.6. The schedule generator requires the information from the schedule queue, data writer and ONNX EPT predictor to generate schedules and determine the best outcome.

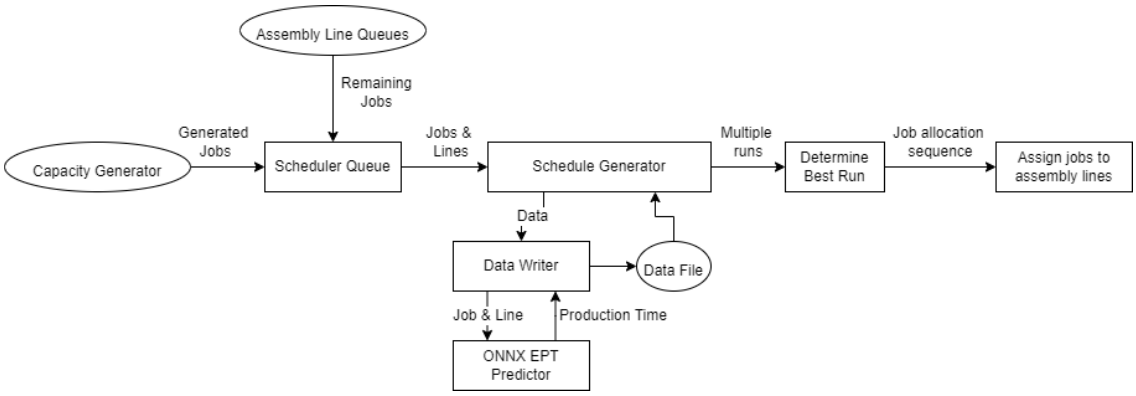


Figure 6.6: Schematic overview of the scheduler

Performance Trackers

Beside the essential model elements, the simulation also contains performance trackers. These trackers are triggered either time-based or event-based and obtain critical information about the simulation performance. The two most important trackers are the KPI tracker and the reschedule tracker, with the latter one being discussed later. A Key Performance Indicator (KPI) value is a quantifiable measure of performance over time for a specific objective. These KPI values are necessary to determine the performance of the simulation and are divided into four categories: throughput, tardiness, state durations and reschedule impact. Each KPI value is evaluated on a weekly interval rate.

The throughput represents the output performance, determined for four different KPI values. First of all, the weekly number of jobs that are processed is tracked. Secondly, the weekly number of dies processed is tracked. The number of dies in a job is determined by the production quantity and the consume factor. Thirdly, the weekly capacity processed is evaluated. This is the workload that has been processed in one week time, determined by the difference between the weekly

maximum capacity and the remaining workload at the end of the week. Finally, the average effective production time is tracked. The EPT, also known as the total machine completion time, is determined by the time it took the assembly line to process the entire job. The average EPT represents the average machine completion time of all jobs processed in that week.

The tardiness represents the performance with respect to the due date and is expressed in four KPI values. First, the expected job tardiness is determined. Tardiness is expressed in the number of days a job is late, only using integer values. If a job is one minute late, the tardiness of that job is one day. If a job is completed on time (before the due date), the tardiness is zero. The expected job tardiness is the average tardiness over all jobs processed that week. The second value is the expected job lateness. Lateness is expressed in the hours a job is late and does not round up any value. The expected job lateness is the average lateness over all jobs processed that week. The third value is the weekly number of jobs late. The last value is the weekly total tardiness, which is the sum of the tardiness of all jobs processed that week.

The state durations help to provide extra information on the system performance, using four KPI values. First, the weekly time spent in setup state is tracked. Secondly, the weekly time spent in production state is tracked. Thirdly, the weekly time spent in down state is tracked. Finally, the weekly time spent in idle state is tracked. These KPI values are determined for each individual machine in an assembly line, expressed in hours.

The reschedule impact represents the impact a rescheduling procedure has on the initial schedule and the resources, reflected in three KPI values. First of all, the number of times rescheduling occurred per week is tracked. The frequency determines the number of times a new schedule is generated. The second value is the average number of jobs rescheduled. This value is determined by taking the sum of all jobs rescheduled in a week and divide it with the rescheduling frequency. The last value is the average number of lines rescheduled, determined by the sum of all lines taken into account during rescheduling and dividing it with the rescheduling frequency.

With these 15 KPI values, critical information on the simulation performance is obtained. The throughput is the most important criterion, followed by the tardiness performance. The state durations mainly helps to provide extra information and the reschedule impact is used to evaluate and compare the rescheduling heuristics later on. All KPI values have been listed below:

- Throughput
  - Weekly number of jobs processed
  - Weekly number of dies processed
  - Weekly capacity processed
  - Average effective process time
- Tardiness
  - Expected job tardiness
  - Expected job lateness
  - Weekly number of jobs late
  - Weekly total tardiness
- State durations
  - Weekly time spent in setup state
  - Weekly time spent in service state
  - Weekly time spent in down state
  - Weekly time spent in idle state
- Reschedule impact
  - Number of times rescheduling occurred
  - Average number of jobs rescheduled
  - Average number of lines rescheduled

Since evaluating and comparing the simulations on 15 KPI values is unfeasible, the four most important KPI values have been determined. First, the weekly number of dies and weekly number of jobs are selected, since the combination of two provides a proper evaluation on the throughput performance. The third value is the expected job lateness, providing the best tardiness performance indication. The fourth chosen KPI value is the weekly idle time, because this is an important indicator on the scheduling performance.

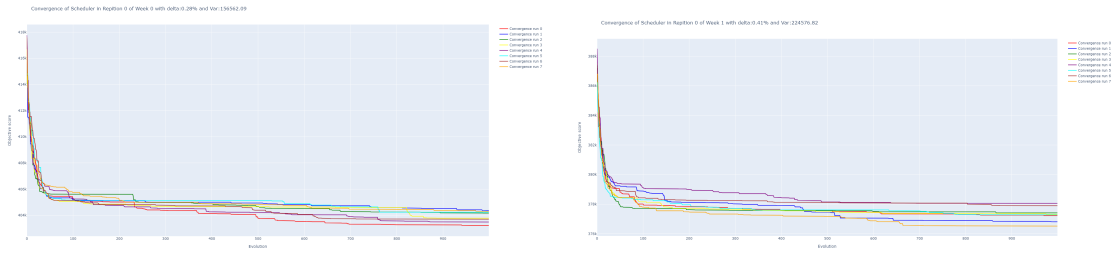
6.3 Simulation Settings

After creating the simulation model, several settings have to be determined. The three general settings are the schedule settings, the simulation length and the required warm up period. Each setting will be elaborated in this section.

6.3.1 Schedule Setting

The schedule settings determine the performance and limitations of the scheduler. One important schedule setting is the algorithm objective function. For this research, the objective function is set to minimize the make span. Furthermore, the scheduler is allowed to iteratively evolve a certain number of generations. The higher the parameter, the higher the number of generations, resulting in a better chance to derive the optimal objective. However, with an increasing number of generations, the required computational time increases as well. Whereas the number of generations can be seen as the length of one schedule execution, the number of replications is set by the number of runs. Each run uses the same data set and tries to solve the same problem. The number of runs determine how many final results are compared. Comparing more runs slightly increases the chance to obtain better results, but the computational time increases significantly with more runs. Therefore, a proper balance should be derived.

In order to determine this balance, the convergence of the objective score is analysed. The scheduler is used for eight runs, each converting the same problem data set. The maximum number of evolutions is set to 1000. The results for two individual weeks are presented in Figure 6.7a and Figure 6.7b. In the figures, eight different runs are presented, each evaluated on their objective score for each derived evolution. In the title of the figures, a delta and variance score are presented. The delta score provides the relative difference in percentage between the best and worst convergence run. The variance is also determined for the best and worst convergence run.



(a) Objective score Week 1

(b) Objective score Week 2

Figure 6.7: Scheduler objective score convergence

Both figures show similar behaviour, with the objective score improving significantly in the first 50 evolutions, followed by a decreasing decline. Around the 200 to 300 evolutions, the objective score stabilizes and has almost no improvement until the last evolution. It is determined that the objective score is feasible after 400 evolutions. The use of eight runs seems redundant, especially since it increases the computational time. The scheduler can execute multiple runs parallel to each other. When testing the number of parallel runs, the best computational time was achieved with four runs parallel. More runs seem to degrade the performance, increasing the computational time significantly. Furthermore, comparing four individual runs of the same data set returns feasible results. Therefore, the scheduler will solve each problem by using four runs with 400 evolutions each.

6.3.2 Simulation Length

Since the simulation uses several stochastic components, it is important to determine the influence of randomness on the results. Stabilizing the randomness can be achieved by either running longer

simulations or more repetitions. It is chosen to focus on longer simulations.

To determine how many weeks of simulation are required, a KPI convergence method is derived. A simulation is executed with a length of 550 weeks. With the convergence method, the results of this simulation are split into 550 entries. The first entry takes the data from the first week, the second entry takes the data from the first two weeks and so on. The last entry will thus take all the data into account. By using this method, the influence of the number of weeks can be determined.

The data is analysed on three aspects: the mean, the confidence interval and the relative confidence interval. The four KPI values as derived in Section 6.2.2 are used to analyse the behaviour. The mean is the average value of the KPI analysed. The convergence of the mean is shown in Figure 6.8.

The confidence interval measures a degree of certainty in a sampling method. It is a range, bounded above and below the mean, that likely contains an unknown population parameter. The confidence level refers to the certainty that the interval contains the true population parameter. By establishing a 95% confidence interval, using the sampled mean and standard deviation, an upper and lower limit are derived that contain the true mean with 95% certainty. The range of the confidence interval is presented, while using the convergence method, in Figure 6.9.

The relative confidence interval is the ratio between the confidence interval and the mean. It expresses the relative range around the mean. The relative confidence interval is shown in Figure 6.10, also using the convergence method.

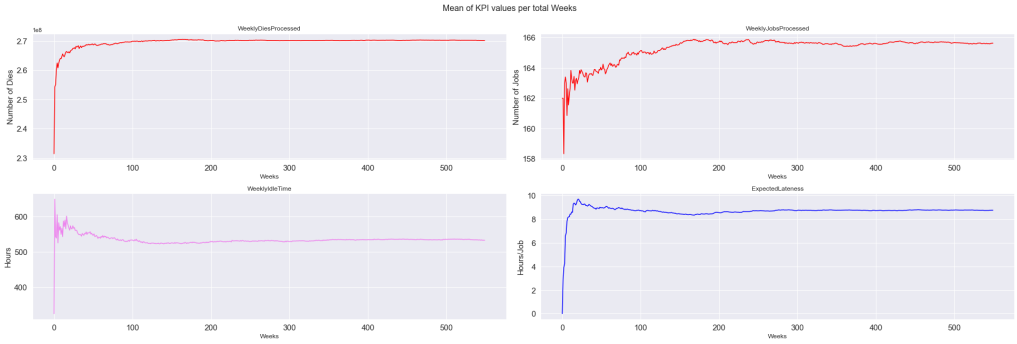


Figure 6.8: KPI convergence Mean

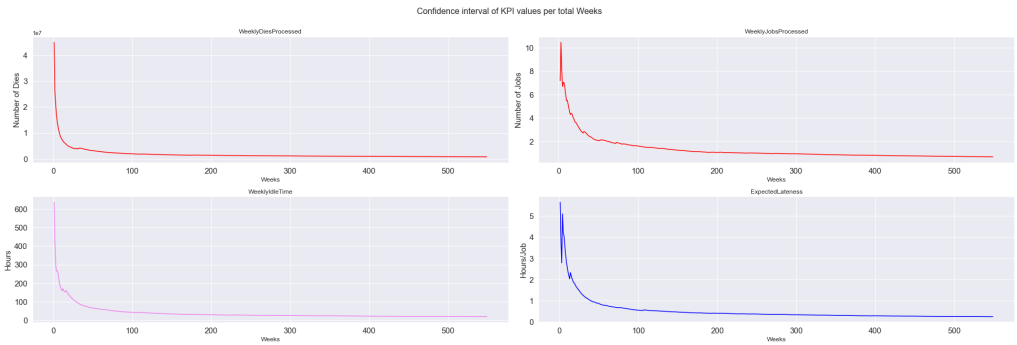


Figure 6.9: KPI convergence Confidence Interval

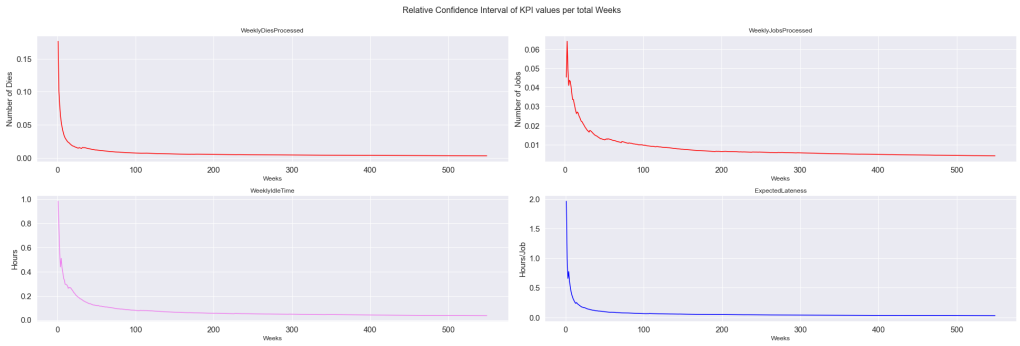


Figure 6.10: KPI convergence Relative Confidence Interval

For the simulation length, the most important value is the relative confidence interval, since this value is most representative for the influence of randomness and has the same accuracy for the different KPI values. If all four values drop below 0.01, it means the next sampled data point varies less than 1% around the mean with a 95% certainty, which is sufficient to call the randomness stabilized. When analysing Figure 6.10, all four values drop below 0.01 after approximately 200 weeks. However, when analysing Figure 6.9, the values still seem to decline slightly after 200 weeks. It is therefore chosen to perform the parameter tuning with 200 weeks and to generate the final results with 400 weeks of simulation.

6.3.3 Simulation Warm Up

The length of a simulation decreases the influence of randomness on average. However, since the simulation starts out empty, the first couple of weeks might deviate too much from the rest, incorrectly influencing the average. Therefore, a warm up period is introduced, which allows the simulation to run but no data is obtained. Therefore, the warm up period should stabilize the starting point compared to the rest of the simulation.

To determine the length of the warm up, Figure 6.8 is the most useable source. It can be seen that most mean values require a couple of weeks to stabilize, but the weekly jobs processed require a lot more weeks due to the large stochastic influence on size and speed. Another plot is created with the mean values using the convergence method, but this time the warm up period is varying. The same set of data is used, but the simulation length is fixed on 300 weeks and the warm up period is altered between 0 and 100 weeks, with a stepsize of 20 weeks. The result is shown in Figure 6.11

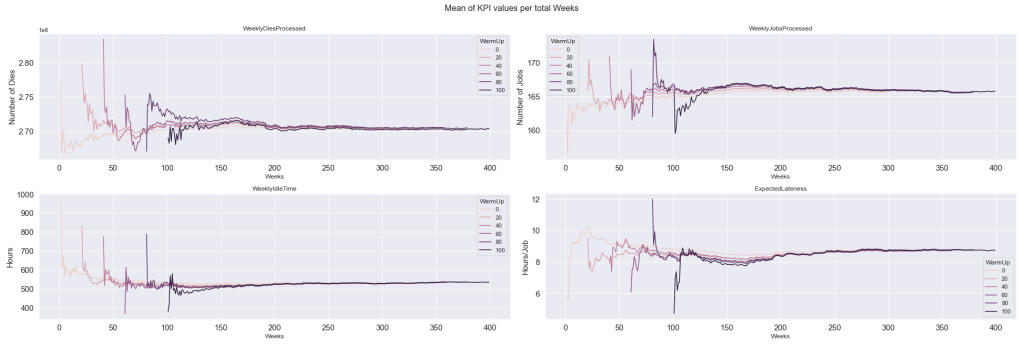


Figure 6.11: KPI convergence with variable warm up period

The figure shows the mean KPI values and how many weeks they require to stabilize. It can be seen that even though the warm up period increases, the same stabilization value for the means is

achieved. Thus, the warm up period has minimal influence but a small warm up period converges faster than no warm up period. It is therefore chosen to use 20 weeks as a warm up period.

The simulation settings have been determined with the scheduler using four runs running parallel, each having 400 evolutions. The simulation length is set to 200 weeks for parameter tuning and 400 weeks for the final results. The simulation warm up period is set to 20 weeks.

6.4 Rescheduling Heuristics

The current model is able to mimic the behaviour of the ATSN factory, which can be used to predict the average performance. However, the focus of this research is with regard to the reschedule heuristics and their influence on the performance. As mentioned in Chapter 3, two questions require an answer: when to reschedule and how to reschedule. The first refers to the trigger that activates the reschedule procedure. Three different rescheduling heuristics have been developed, each approach the when to reschedule with a different tactic. The how to reschedule refers to how the rescheduling is conducted. It is chosen to do this either fully or partly.

One of the performance trackers in the model is the reschedule tracker. This tracker validates the system with an interval of 30 minutes. If the tracker determines that the heuristic should be triggered, the rescheduling procedure is initiated.

6.4.1 Time Based

The first rescheduling heuristic is called *time based*. This heuristic is time driven and requires no parameter to evaluate the system. Because no parameter is required, this heuristic is completely independent of the system performance. A time interval is determined, based on the input variable H, as shown in Equation (6.11).

$$T_{int} = \frac{H+1}{7} \tag{6.11}$$

T_{int} represents the time interval and H represents the number of times rescheduling is required. If H would be zero, no rescheduling would occur, and a new schedule is generated at day 0 of each week. If H is set to one, a new schedule will be generated midway of the week. The higher H becomes, the shorter the time interval, the more often rescheduling will occur. If H would be equal to six, rescheduling would happen at the end of each day. A visual representation of the heuristic is shown in Figure 6.12.

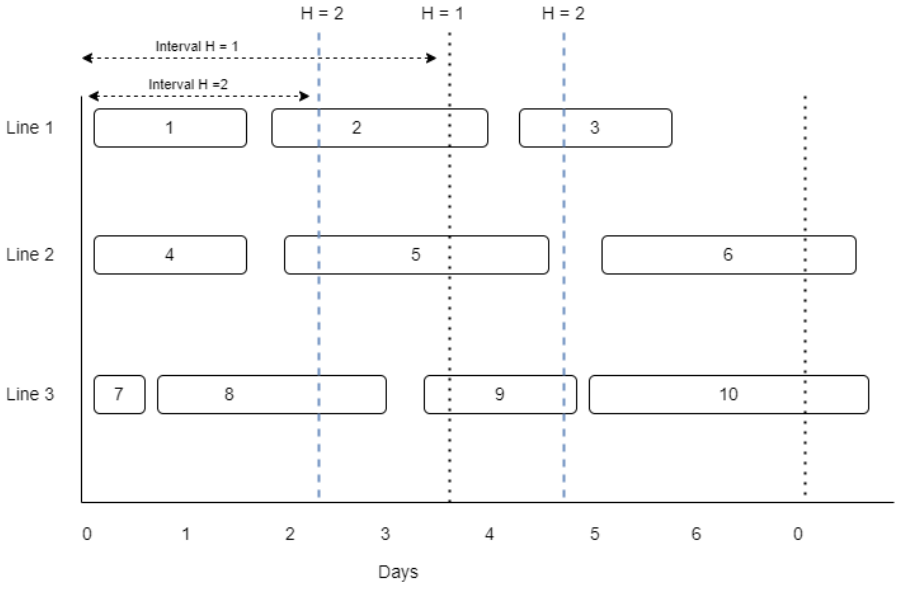


Figure 6.12: Time based heuristic

The *time based* heuristic will only be used with full rescheduling. In other words, all lines are taken into consideration when rescheduling occurs.

6.4.2 Capacity Balance

The second rescheduling heuristic is called *capacity balance*. This heuristic is event driven and evaluates the system on the capacity balance. The capacity balance is determined for each individual line using Equation (6.12).

$$CB_{line} = T_{left} - (\sum_{i=0}^{i=J-1} T_{p_i} + T_{RJ}) \tag{6.12}$$

CB_{line} is the capacity balance of the line, with J the current number of jobs in the queue. T_{left} is the time remaining in the current week and T_{p_i} is the predicted duration for job i in the queue, with $i = J$ being the job in process. Finally, T_{RJ} is the predicted effective production time remaining for the job in process.

Since no down time is considered in the predicted duration, the capacity balance will always be positive at the start of the week. The more time passes, the smaller T_{left} will become and hopefully the predicted capacity decreases similarly. However, all time spent in down state prevents the capacity from decreasing and results in a stationary capacity workload. When the capacity balance becomes negative, the predicted required capacities become more than the time left in the week. In other words, when the capacity balance is positive, it is expected that the assembly line will become idle by the end of the week. If the capacity balance is negative, it is expected that the assembly line will not finish its workload and jobs will be late. A visual representation of the heuristic is provided in Figure 6.13

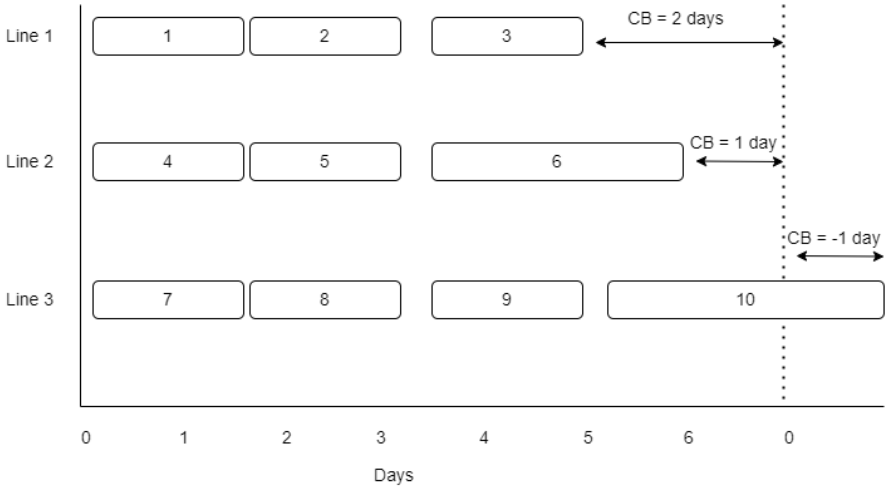


Figure 6.13: Capacity balance heuristic

All heuristics can be used for full rescheduling. However, since the *capacity balance* heuristic is event driven, it uses a parameter to evaluate the system, in this case the capacity balance. Therefore, it is possible to apply partial rescheduling to this heuristic as well. With fully rescheduling, the heuristic evaluates the average capacity balance of all active lines. Active lines represent all assembly lines that are not idle, including lines that are down. By taking the average capacity balance, the heuristic evaluates the overall performance of the factory. In Figure 6.13, the average capacity balance is equal to 0.67 days. When the average capacity balance becomes smaller than a threshold, the system reschedules with all lines considered.

For partial rescheduling, the performance of each individual line is evaluated. By using Equation (6.12), the capacity balance of each individual line is retrieved. A line can be classified as a bad line, good line or not of interest. A positive and negative threshold value are set to classify the lines. Bad lines represent the assembly lines that have a negative capacity balance, that is past the negative threshold value and thus require rescheduling. Good lines have a positive capacity balance larger than the positive threshold value and feature space to obtain more jobs. All lines in between the two threshold values are performing sufficient and are not of interest. If there is at least one good and one bad line, partial rescheduling is triggered.

6.4.3 End Time Drift

The last heuristic is called *end time drift*. Whenever a schedule is generated, the jobs are allocated to specified assembly lines in a determined sequence. Based on the predicted duration and the current time, each job gets a scheduled end time assigned. This scheduled end time is the time the job is finished if no interference occurs. However, since the predicted duration does not take down time into account, the scheduled end time will gain drift each time a job is not processing, i.e., the assembly line is in down state. Therefore, the drift is the difference between the scheduled end time of the job and the current time of the simulation. If the scheduled end time is in the future, the drift will be negative and will be neglected, i.e., set to zero. If the scheduled end time is in the past, the drift will be positive and will be taken into account.

For fully rescheduling, the average drift of the active lines is taken. If this average drift becomes larger than a threshold value, rescheduling is initiated. A visual representation of the heuristic is provided in Figure 6.14, where the average drift is equal to 0.50 days per line.

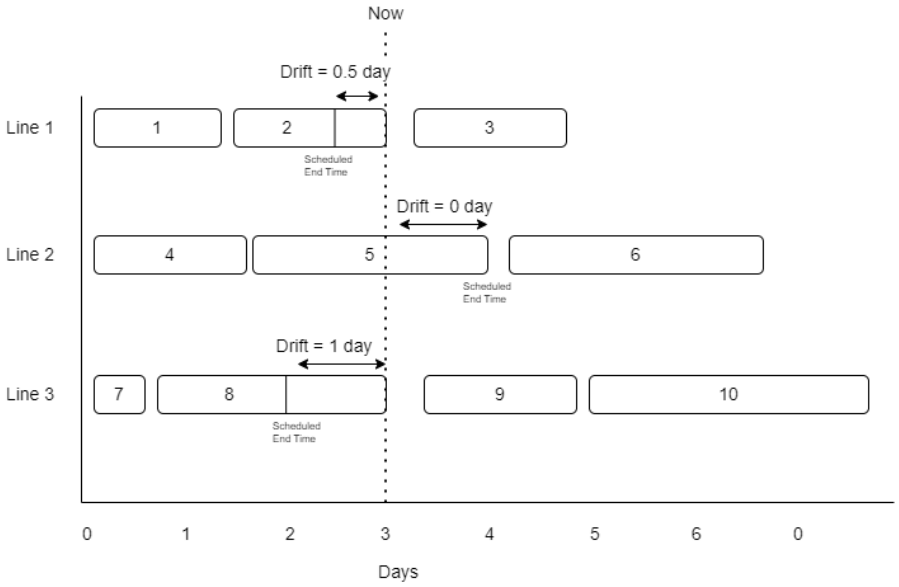


Figure 6.14: End time drift heuristic

For partial rescheduling, each individual line is evaluated on their drift performance. If the drift becomes larger than the bad threshold value, the line is classified as a bad line. Since a negative drift is not representative for the performance, the good lines will be determined with the capacity balance, as explained before. When there is at least one good line and one bad line classified, partial rescheduling is triggered.

Chapter 7

Results

With the model complete, results can be generated and analysed. First of all, the result of the simulation without rescheduling will be compared to the historical data. Afterwards, the parameters for the different heuristics will be screened and tuned. Based on the tuning, the best results with respect to throughput and tardiness will be selected and compared. Based on those best results, two different simulation environments are tested.

7.1 Simulation Validation

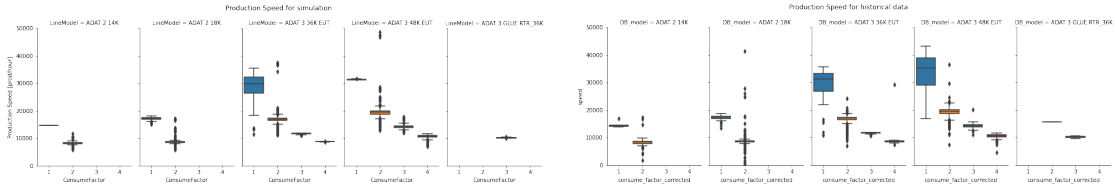
To understand whether the simulation generates valid results compared to the historical database, a validation is required. There are multiple important aspects to validate, but the most important ones have been selected and will be elaborated next.

7.1.1 Machine Processing Speed

One of the stochastic parameters used is the machine processing speed. As explained in Subsection 6.2.2, the speed is sampled from a subpopulation of the historical data, based on the consume factor and the die bonder model. For each machine, a production speed is sampled and the assembly line production speed is the sum of the sampled machine production speeds. This speed is validated against the historical data to determine the influence of the stochastic behaviour.

There are two types of speed that require validation. The first one is the production speed, which is the processing speed without taking environmental factors such as setup and down times into account. The second validation is aimed at the effective production speed, which takes the entire order duration into consideration. To validate the speeds, the observations from the historical database, as derived in Chapter 5, are compared to the observations of a simulation of 400 weeks with 20 weeks warm up period.

The results for the production speed are plotted in Figure 7.1. The figure shows how the sampled speeds are distributed per subpopulation by using the boxplot method. The center line represents the median of the data, with the box showing the first and third quarter, thus 50% of the data. The outer lines indicate the spread of the remaining data with the dots indicating outliers. On the vertical axis, the number of products produced per hour is shown. On the horizontal axis, the consume factor is shown. The five plots represent the five different machine models.

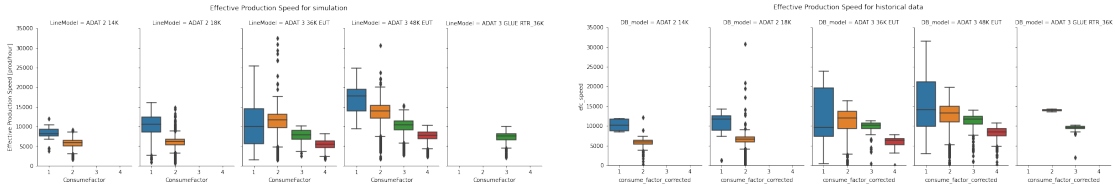


(a) Simulation data (b) Historical data

Figure 7.1: Effective process speed validation

It should be noted that the historical data has almost 16.000 entries, each representing an individual machine observation. Meanwhile, the simulation data includes 66.000 entries, each entry representing an assembly line observation, with the production speed being the average production speed over the number of machines in the assembly line. The two data sets appear to have similar median values and the spread is correctly mimicked. However, the subpopulations of the historical data with only a few observations, as presented in Table 6.3, show different spreads due to insufficient data. Furthermore, the outliers of the simulation data seem to be less divergent than the historical data outliers. Nonetheless, the distribution of the data appears to behave similar. A more extended validation is presented in Section A.1, where each subpopulation is numerically evaluated on the similarity. The extended validation concluded that the average weighted ratio between the simulation data and the historical data is 0.999 for the production speed. It can therefore be concluded that the production speed is correctly implemented.

The results for the effective production speed are plotted in Figure 7.2. The same method is used as before but now the effective production speed is determined by taking the total duration of the order. The similarity in distribution is again high, but the simulation data has more spread due to the implementation of the down ratio. Nonetheless, the distribution and the median values appear to behave similar, thus the implementation of the down ratio is correct. According to the extended validation, the average weighted ratio between the simulation data and the historical data is 0.980 for the effective production speed. This is explained in Section A.2.



(a) Simulation data (b) Historical data

Figure 7.2: Average process speed validation

7.1.2 Expected and predicted duration accuracy

During the simulation, a total of three different duration values are assigned to the job: the expected duration, predicted duration and realized duration. The expected duration is determined as explained in Subsection 6.2.2 and is used for the capacity generator. This duration is dependent on the average process speed of the device type, the mean down time and the production quantity. Based on this duration, the capacity generator determines how many jobs can be loaded into the system. If this value deviates too much from the realized duration, the workload of the system becomes too high or too low.

The predicted duration is determined by the ONNX machine learning model. This prediction only includes the production time and the setup time. It is used for the scheduler and to determine the predicted workload for the assembly lines. The predicted duration should be more accurate

than the expected duration, because it already takes the assembly line attributes into account, along with multiple order attributes.

The realized duration is determined by the length of the production event, equivalent to the sum of the setup time, the sampled production time and the sampled down time.

The accuracy of the expected duration determines the resemblance to the realized duration. If the expected duration of a job is equal to the realized duration, the expected accuracy is 100%. If the expected duration of a job was two times larger than the realized duration, the expected accuracy is 200%.

The accuracy of the predicted duration determines the resemblance to the realized production time and setup time, since down time is not considered. If the predicted duration of a job is equal to the sum of the realized production time and setup time, the predicted accuracy is 100%. If the accuracy of the predicted duration is 50%, the sum of the production time and setup time is twice as large as the predicted duration.

For a simulation of 400 weeks, with 20 weeks of warm up period, both the accuracy of the expected and prediction duration are validated. The result is shown in Figure 7.3. The left plot shows the accuracy of the expected duration and the right plot shows the accuracy of the predicted duration.

It can be seen that the predicted accuracy is completely centered around the 100%, meaning that the prediction is really accurate for all jobs. The mean accuracy is 97.87%. The expected duration has more spread, meaning some predictions are too high and some are too low. It can be seen that the highest peak is centered around 90% and the second largest peak is centered around 50%, meaning the realized duration is two times larger than the expected duration. Nonetheless, the mean accuracy is 101.15% identical to the realized duration, which is an overall good performance. Thus, it can be concluded that both predictions are correctly implemented, showing an accurate understanding of the system behaviour, resulting in valid predicted values.

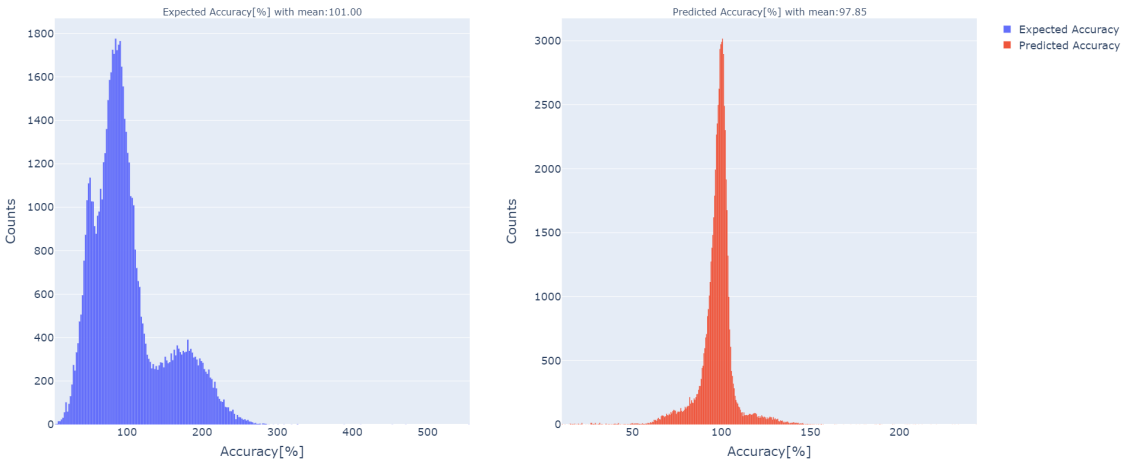


Figure 7.3: Expected and predicted duration accuracy

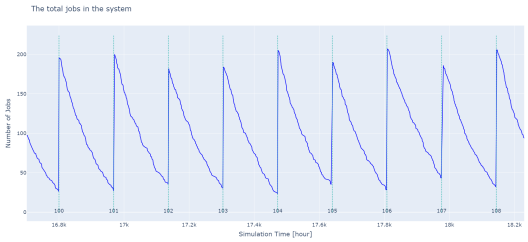
7.2 Baseline performance

The baseline is the simulation without any rescheduling heuristic. It is used to mimic the real-world behaviour and determine the gain achieved by any rescheduling heuristic. The baseline simulation is ran for 400 weeks with a warm up period of 20 weeks. The most important aspects to analyse will be elaborated.

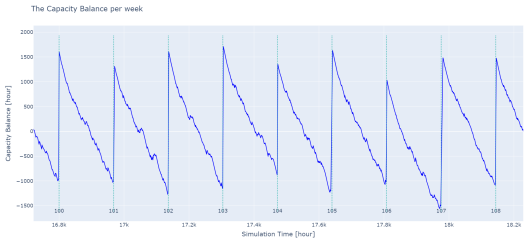
First of all, processing jobs is one of the main goals of the simulation. In Figure 7.4a, the total number of jobs in the system is plotted against the simulation time. A subset of the total data

is taken to show the behaviour, representing week 100 to 108. It can be seen how the number of jobs declines during the week and is refilled at the start of each week. It can also be noted that the number of jobs never reaches zero, which causes tardiness for jobs.

The same subset of data is taken for the capacity balance, shown in Figure 7.4b. At the start of the week, the system predicts it can produce all the required workload, with more than a thousand spare hours over all assembly lines. This means that, on average, each assembly line has one day of spare capacity left. If everything would be produced as predicted, the capacity balance would be a stationary line. Nonetheless, due to the down times and stochastic production times, the balance degrades during the week. At the end of the week, the capacity balance is approximately a thousand hours negative, which means that on average each assembly line has more than one day of workload left. This is confirmed by Figure 7.4a, since jobs remain in the system.



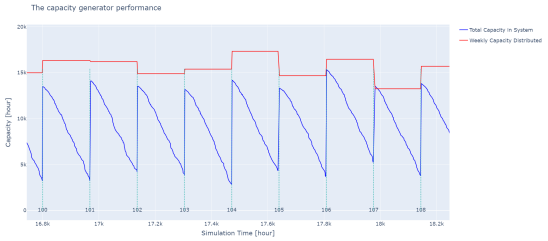
(a) Total jobs in the system



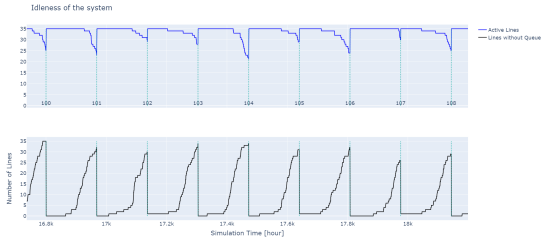
(b) Capacity balance of the system

Figure 7.4: Baseline throughput performance

The remaining workload left at the end of the week influences the weekly capacity that is generated at the start of the next week. The capacity generator determines the *weekly maximum capacity* by using the workload remaining in the system and the idle time of the previous week, as shown in Equation (6.10). In Figure 7.5a, the performance of the capacity generator is shown. The vertical axis shows the total capacity expressed in hours, whereas the horizontal axis is the time in the simulation. The blue line represents the capacity in the system over time. The red line indicates weekly capacity that is distributed, which is adapted per week. The idleness is shown in Figure 7.5b, where the blue line indicates the number of active lines and the black line indicates assembly lines with an empty queue. It can be seen that at the end of every week, several assembly lines are not producing anymore. This is caused by the fact that during the week, some lines already have empty queues and thus no job waiting when the line is finished.



(a) Capacity generator performance



(b) Idleness performance of the system

Figure 7.5: Capacity and idleness performance

As explained in Subsection 6.2.2, the total time of a job is determined by the setup time, the production time and the down time. It varies per job how these durations are balanced. By taking the ratio of each duration of the total time, the contribution of each duration to the total time can be determined. The result is shown in Figure 7.6. The left plot (blue) shows the distribution of

the production ratio, determined by dividing the production time by the total time. The middle plot (red) shows the distribution of the down ratio. The right plot (green) shows the distribution of the setup ratio, determined by dividing the setup time by the total time.

It is noticed that the setup ratio has a minimal contribution to the total time. The distribution of the production ratio and down ratio seem to be inverted towards each other, with the production ratio being left skewed and the down ratio right skewed.

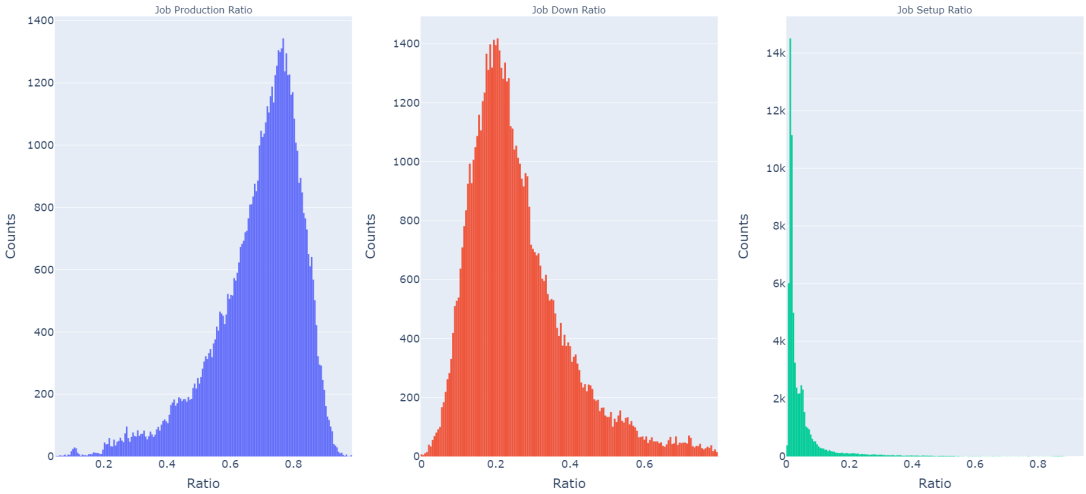


Figure 7.6: The ratio distribution of the different durations

7.3 Parameter tuning

In Section 6.4, the rescheduling heuristics have been elaborated. A total of six heuristic parameters have to be tuned to optimize the heuristics. Before the parameter tuning, a broader approach was taken to determine the influence of several parameters to limit rescheduling behaviour. However, their influence was deemed negligible and the focus was aimed at the following six parameters. Notice that the *capacity balance* and *end time drift* heuristics share one common parameter, which is tuned for both heuristics individually.

- Time Based
  1. The number of times rescheduling is required per week
- Capacity Balance
  1. The threshold capacity value for the average capacity balance
  2. The threshold capacity value to determine a bad line
  3. The threshold capacity value to determine a good line
- End Time Drift
  1. The threshold drift value for the average drift
  2. The threshold drift value to determine a bad line
  3. The threshold capacity value to determine a good line

The parameters will be evaluated against the KPI values as explained in Section 6.2.2. As stated before, it is infeasible to derive conclusion while comparing all 15 KPI values. Therefore,

four KPI values have been chosen to evaluate the performance. These KPI values include the weekly number of jobs and weekly number of dies processed, representing the throughput. The expected job lateness, evaluating the tardiness performance. The idle time, evaluating the possible gain through rescheduling. Furthermore, it is chosen to evaluate all reschedule impact KPI values, where the average number of lines and the average number of jobs are combined under the average resources rescheduled.

Each heuristic will be compared against the same baseline, where no rescheduling is performed. The baseline is indicated with 0 in all figures below. The other numbers on the horizontal axis are the combinations of parameters for each rescheduling heuristic.

7.3.1 Time Based

For the *time based* heuristic, it is chosen to vary between one to ten times rescheduling per week. Aside from the rescheduling frequency, no other input parameter is required. The results can be found in Figure 7.7.

The simulation on the upmost left is the baseline performance, indicated with 0. The second result is from the simulation where $H = 1$, so a rescheduling frequency of one. This frequency increases for every simulation to the right. The utmost right simulation uses a rescheduling frequency of 10, thus generating 11 schedules per week.

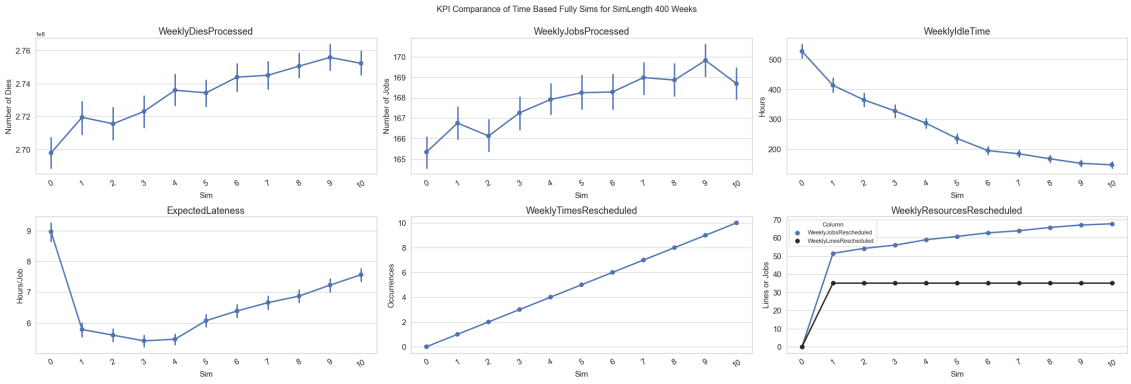


Figure 7.7: Time based results

Figure 7.7 presents the six KPI values to determine the performance. The number of dies is scaled with $1e^8$ and is expressed in dies, just like the number of jobs is expressed in jobs. The idle time is expressed in hours over all machines and the expected lateness is the average hours a job is late. The last two values represent the number of times rescheduling occurred per week and the number of resources rescheduled on average, with the jobs and lines considered. All values are determined on a weekly basis with the figure showing the average and confidence interval of those 400 mean values.

It can be seen in Figure 7.7 that the more often rescheduling occurs, the higher the weekly throughput. It almost appears to be a linear increment, whereas the difference between no rescheduling and nine times rescheduling is 2.15%. The increase in throughput is reflected in the decrease in idle time, which means that the machine utility rate is maximized with a higher rescheduling frequency.

Where the throughput performance increases with the number of rescheduling occurrences, the expected lateness performance decreases. The difference with no rescheduling and nine times is still an improvement of 19.38%. However, the tardiness performance decreases when rescheduling occurs more than three times per week. No conclusion can be drafted on this behaviour, but the scheduler objective is focused on maximizing the throughput by minimizing the maximum completion time. Since this objective has no optimizing criteria regarding the tardiness, rescheduling

later in the week might cause jobs to be more frequently late, thus decreasing the tardiness performance. The optimal tardiness performance is achieved with three times rescheduling per week, with an improvement of 39.60%.

The number of resources rescheduled with the *time based* heuristic is high. When the rescheduling frequency is equal to nine, the total number of jobs rescheduled per week is 603, with 35 lines taken into consideration. For practical implementation, this might be problematic.

For practical implementation, it is more interesting to look at a lower rescheduling frequency, since the impact on the factory is more feasible. With a rescheduling frequency of one, the number of jobs rescheduled per week is 51 on average. Meanwhile, the throughput increases with 0.80% and the tardiness performance improves with 35.48%.

7.3.2 Capacity Balance Fully

For the *capacity balance fully* heuristic, there is one parameter that needs to be optimized. This is the threshold capacity value with respect to the average capacity balance. In other words, whenever the average capacity balance becomes less than the threshold value, rescheduling is triggered.

The heuristic is first screened for the threshold value between zero and one, where the value is expressed in negative days times the number of active lines. In other words, when the threshold value is equal to 0.5, the average capacity balance has to be more than -0.5 days for all active lines to trigger rescheduling. The performance decreases quickly when the threshold value is higher than 0.4. Therefore, parameter tuning is performed for the lower values. The result can be seen in Figure 7.8.

The simulation result on the left, indicated with 0, is the baseline performance. The other three values represent the simulations where the threshold value is -0.0, -0.2 and -0.4 days for the average capacity balance.

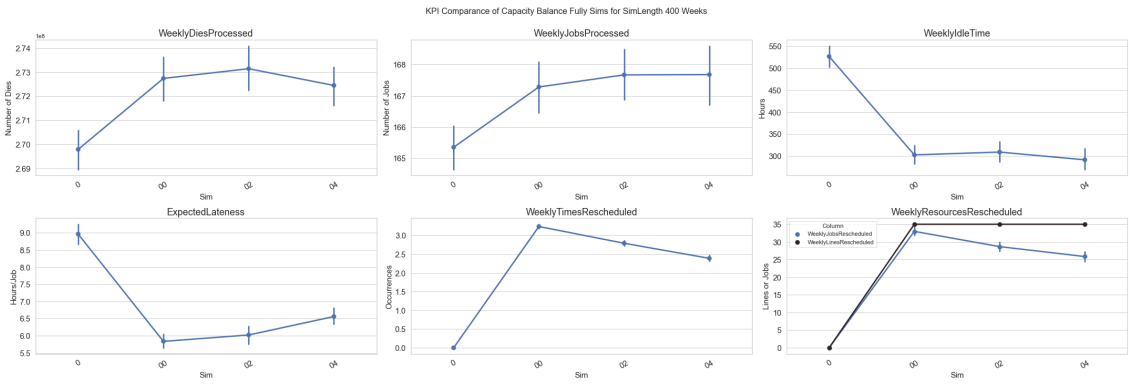


Figure 7.8: Capacity balance fully results

It can be seen that the best throughput performance is achieved with a threshold value of 0.2 days. The weekly number of dies processed increases with 1.24%. Furthermore, the number of rescheduling occurrences and the number of jobs taken into consideration is much lower compared to the results of the *time based* heuristic. Figure 7.8 clearly shows that with a higher threshold value, rescheduling occurs less often and the average number of jobs decreases as well.

The best tardiness performance is achieved with decreasing the threshold value to 0.0 days. This means that whenever the average capacity balance becomes negative, rescheduling is triggered. The expected lateness per job is improved with 34.89% compared to the baseline.

7.3.3 Capacity Balance Partly

The *capacity balance partly* has two parameters that require tuning. The first one is the threshold capacity value to determine bad lines. The second one is the threshold capacity value to determine good lines. During screening the influence of the number of lines required to reschedule was analysed, but the difference between requiring one or more good and bad lines was negligible.

The good line threshold value was analysed between -0.2 days and +0.3 days, with a stepsize of 0.1 day. In other words, whenever the capacity balance of an individual line is lower than the -0.2 days threshold value, the line is classified as good line. Similarly, the bad line threshold value varies between -0.7 and -0.9 days, with a stepsize of 0.1 day. Any line with a capacity balance exceeding the -0.7 days threshold value is classified as bad line. The results are shown in Figure 7.9.

The simulation result on the left, indicated with 0, is the baseline performance. Next, the simulation result for the parameter combination of -0.2 days and -0.7 days is shown. The good line threshold value is increased with 0.1 day for each simulation to the right. The bad line threshold value is decreased with 0.1 day for each two simulations to the right.

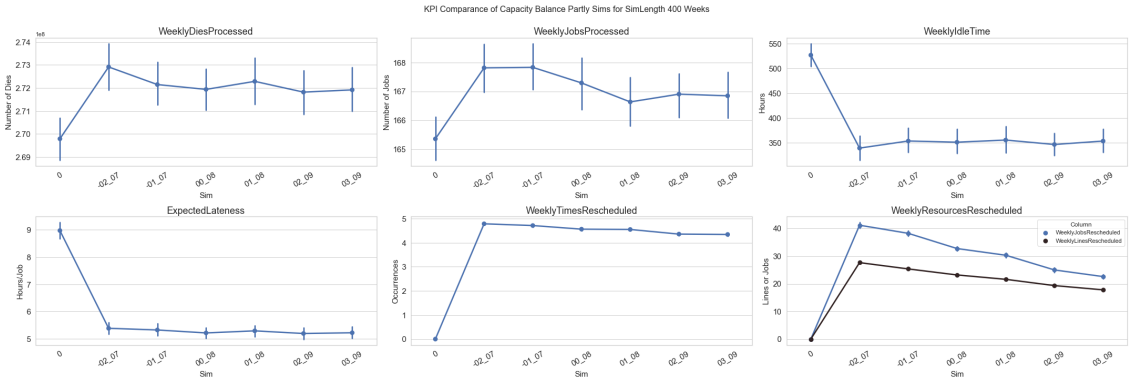


Figure 7.9: Capacity balance partly results

When analysing the results, it appears that the best throughput performance is achieved with both threshold values at their minimum. The results for -0.2 days and -0.7 days show an improvement of 1.16% in the weekly dies processed. The throughput performance seems to decrease when the threshold values increase.

However, the frequency of rescheduling and the resources taken into account also decrease with higher threshold values. The difference between the lowest threshold values and highest threshold values is 0.41% (1.16% to 0.75%) for throughput. At the same time, the amount of resources rescheduled per week decreases from almost 200 jobs for the combination of -0.2 days and -0.7 days to 110 jobs for the combination of 0.3 days and -0.9 days.

It appears that the best tardiness performance is achieved with the highest threshold values. With 0.2 days and -0.9 days for the good and bad lines respectively, the tardiness is improved with 41.98%. The difference in tardiness performance between the different threshold values is minimal.

7.3.4 End Time Drift Fully

For the *end time drift fully* rescheduling heuristics, one parameter needs to be optimized. This parameter represents the threshold value with respect to the average drift and the number of active lines. Take note that only positive drifts contribute to the average. The threshold value is tuned between 0 and 1 day, with a step size of 0.2. The results are shown in Figure 7.10.

The utmost left results represents the baseline, indicated with 0. The second simulation is the result of the heuristic using the threshold value of 0.0 days. For each simulation to the right, the threshold value is increased with 0.2 days.

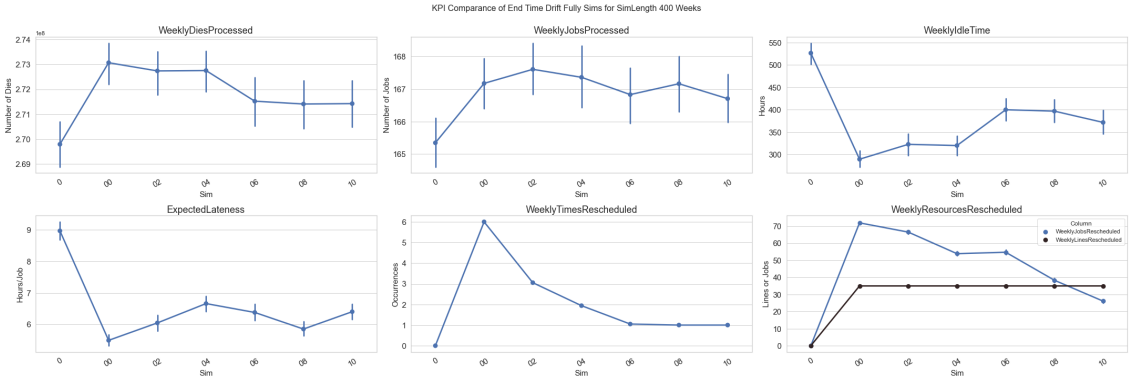


Figure 7.10: End time drift fully results

The best throughput performance is achieved with a threshold value of 0. This threshold is triggered whenever any of the jobs obtains drift and thus reschedules every day. The improvement in throughput is 1.22%. This threshold value also has the best tardiness performance with an improvement of 38.72%. It appears that the throughput performance declines when the threshold value increases.

At the same time, the frequency of rescheduling and the involved number of resources decline when the threshold value is increased. The total resources rescheduled per week decreases from 431 to 200 when the threshold value is increased to 0.2 days. With this threshold value, the throughput gain is 1.09% and the tardiness improves with 32.56%. For practical implementation, this result might be more feasible.

7.3.5 End Time Drift Partly

The *end time drift partly* heuristic has two parameters determining the threshold values for the good and bad lines. The good lines are determined similarly as with the *capacity balance partly*. The bad lines are now evaluated against the drift of each individual assembly line. Whenever the drift becomes larger than the threshold value, the line is classified as bad line. The range for the good line threshold is between 0.00 and 0.75 days, with a stepsize of 0.25 days. The bad line threshold value varies between 0.00 and 0.50 days, with a stepsize of 0.25 days. The results are shown in Figure 7.11

The baseline is presented on the left, indicated with 0, followed by the simulation result with both threshold values set to 0.00 days, indicated with 00_00. The next simulation has the good line threshold value set to 0.00 days and the bad line threshold value set to 0.25 days, indicated with 00_25. For each simulation to the right, one of the threshold values is increased, indicated on the horizontal axis.

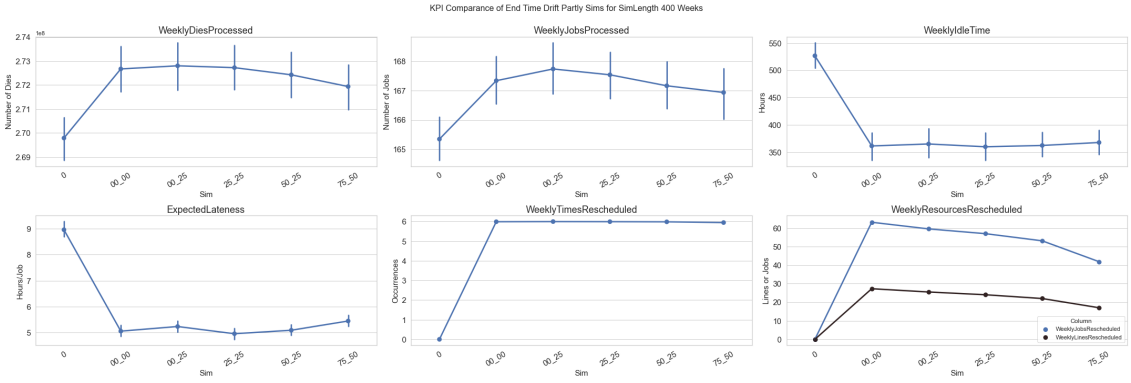


Figure 7.11: End time drift partly results

When analysing the throughput performance, it seems that higher threshold values slightly decrease the performance. The highest number of dies processed is achieved with the threshold for good lines equal to 0.00 days and for the bad lines 0.25 days. Compared to the baseline, the throughput increases with 1.12%.

It can be seen that the number of times rescheduling occurs is maximized, since all event driven heuristics are limited to reschedule only once per day. The resources taken into account are high, with the lowest threshold values rescheduling a total of 357 jobs per week. The number of resources do decline when the threshold values increase.

For the tardiness performance, the difference between the threshold values is minimal. The best performance is achieved with both threshold values equal to 0.25, where the expected lateness is improved with 44.70%.

7.4 Best Throughput Heuristics

In the previous section, all heuristics have been analysed and the best performing threshold combinations have been derived. For the factory, it is critical to improve the throughput. Therefore, the best result for each heuristic is shown in Table 7.1. For the *time based* heuristic, the maximum number of rescheduling occurrences is set to six, since the other heuristics are also limited to rescheduling only once per day.

In the table, the average KPI values are determined over 400 weeks of simulation. Aside of the average KPI value, the confidence interval of 95% is shown. Furthermore, the gain with compared to the baseline is provided. The reschedule influence shows the frequency, the average number of jobs rescheduled and the average number of lines taken into account.

Table 7.1: Best results for throughput performance

Heuristic	Dies Processed [Dies (1e8)]	Expected Lateness [Hour / Job]	Reschedule Influence [Times, avg. Jobs, avg. Lines]	Threshold A	Threshold B
None	2.698 ± 0.009	8.964 ± 0.299	0.0 ; 0.0 ; 0.0	X	X
Time Based	2.744 ± 0.009 [+ 1.71%]	6.387 ± 0.205 [- 28.75%]	6.0 ; 62.7 ; 35.0	6	X
Capacity Balance Fully	2.731 ± 0.009 [+ 1.24%]	6.021 ± 0.246 [- 32.82%]	2.8 ; 28.7 ; 35.0	0.20	X
Capacity Balance Partly	2.729 ± 0.010 [+ 1.16%]	5.391 ± 0.227 [- 39.86%]	4.8 ; 41.1 ; 27.6	- 0.20	- 0.70
End Time Drift Fully	2.731 ± 0.009 [+ 1.22%]	5.493 ± 0.185 [- 38.72%]	6.0 ; 71.9 ; 35.0	0.00	X
End Time Drift Partly	2.728 ± 0.010 [+ 1.12%]	5.235 ± 0.299 [- 41.60%]	6.0 ; 59.5 ; 25.5	0.00	0.25

A visual representation of results is shown in Figure 7.12. The best throughput performance

is achieved with the *time based* heuristic using a rescheduling frequency of six. This heuristic improves the number of dies processed with 1.71%. The difference between *time based* and the other heuristics is almost 0.50%. Between the other heuristics, the difference is almost negligible.

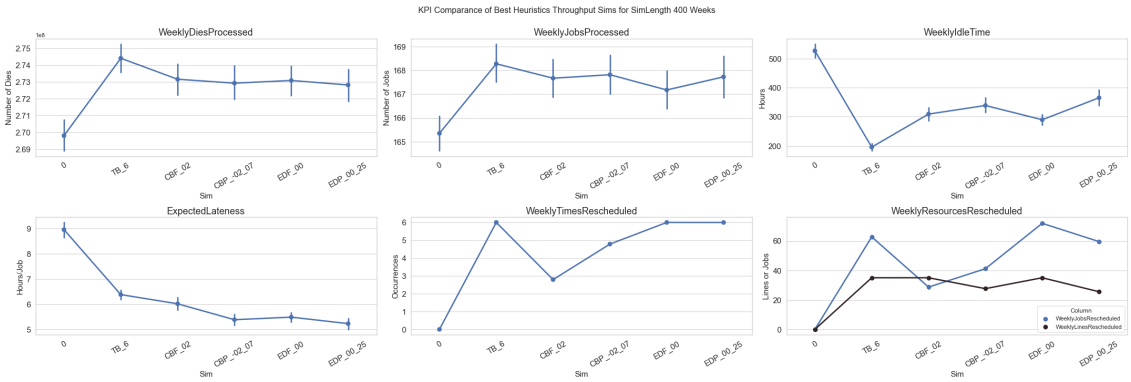


Figure 7.12: Best results throughput performance

It appears that fully rescheduling has the best throughput performance. Compared to partial rescheduling, almost 0.1% extra is gained. For practical implementation, *capacity balance fully* might be the most interesting. This heuristic reschedules with the lowest frequency, takes the least number of jobs into account and has the second highest throughput.

7.5 Best Tardiness Heuristics

The heuristics are again evaluated, but this time with respect to their best tardiness performance. All results are gathered in Table 7.2. The visual representation is shown in Figure 7.13.

Table 7.2: Best results for tardiness performance

Heuristic	Dies Processed [Dies (1e8)]	Expected Lateness [Hour / Job]	Reschedule Influence [Times, avg. Jobs, avg. Lines]	Threshold A	Threshold B
None	2.698 ± 0.009	8.964 ± 0.299	0.0 ; 0.0 ; 0.0	X	X
Time Based	2.723 ± 0.010 [+ 0.93%]	5.414 ± 0.193 [- 39.60%]	3.0 ; 55.9 ; 35.0	3	X
Capacity Balance Fully	2.727 ± 0.009 [+ 1.09%]	5.836 ± 0.208 [- 41.98%]	3.2 ; 33.0 ; 35.0	0.00	X
Capacity Balance Partly	2.718 ± 0.010 [+ 0.75%]	5.200 ± 0.205 [- 41.98%]	4.4 ; 25.0 ; 19.3	0.20	- 0.90
End Time Drift Fully	2.731 ± 0.009 [+ 1.22%]	5.493 ± 0.185 [- 38.72%]	6.0 ; 71.9 ; 35.0	0.00	X
End Time Drift Partly	2.727 ± 0.009 [+ 1.09%]	4.957 ± 0.200 [- 44.70%]	6.0 ; 57.0 ; 24.0	0.25	0.25

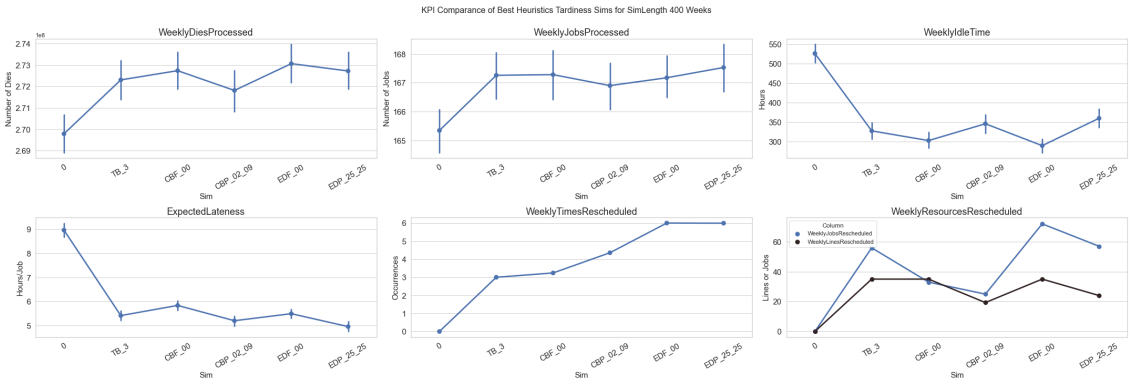


Figure 7.13: Best results tardiness performance

The best tardiness performance is achieved with the *end time drift partly* heuristic, with the threshold values both equal to 0.25 days. The expected lateness is improved with 44.70% compared to the baseline.

Whereas the fully rescheduling heuristics seem to perform better regarding the throughput, the partial rescheduling heuristics achieve the best results for tardiness, although the difference is minimal.

7.6 Different simulation environment

The current simulation is valid, accurate and achieves proper results and gains. However, during the process, choices had to be made in order to create the simulation. For some of the choices, it is interesting to see what would happen if the implementation is adapted. This section describes the different tested simulation environments and elaborates on the results.

7.6.1 Daily Job Generation

The current simulation environment is based on a weekly time schedule. At the start of each week, jobs are generated and dispatched to the system and as a result a new schedule is generated. This behaviour mimics the current implementation of the real-world environment. Nevertheless, jobs arrive at a higher frequency but are not available until the next week. This limitation of available jobs might influence the effects of the different heuristics and is interesting to investigate. Therefore, the simulation is adapted and the capacity generator dispatches jobs on a daily interval.

At the start of the week, the system determines the maximum capacity as explained before. Nevertheless, the capacity generator now determines each day if there is still capacity available in the system. The jobs generated receive a due date for the end of the next week and are enqueued in a waiting queue. Each time the scheduler is activated, the workload in the system is first retrieved. Based on the time left in the week, the system is filled with the extra jobs.

This environment is tested with the best heuristics for throughput and for tardiness. The results are shown Figure 7.14 and Figure 7.15. The first thing noticed is the larger difference between the rescheduling heuristics and the baseline. For the throughput and tardiness performance of the baseline, the difference with the normal environment is negligible. However, the difference between the baseline and the best throughput performance, achieved by *end time drift partly*, is now 2.39% instead of 1.12%. Thus, with a higher job arrival rate than once per week, the throughput performance is better.

The tardiness performance seems to degrade with the higher job arrival rate. The *time based* heuristic has the worse tardiness performance, deteriorated with 22.22% compared to the baseline. The other heuristics seem to behave similarly to the baseline. Only the *end time drift fully* heuristic seems to perform better with respect to the tardiness, even though its performance decreases from

5.493 hours per job to 6.369 hours per job when switching from weekly to daily job arrival rate. This is equivalent to a loss of approximately 10% compared to the baseline.

The heuristics for optimal tardiness performance seem to achieve equal or higher throughput results than the throughput heuristics when using the daily job arrival environment. Similar behaviour can be seen around the tardiness performance of the different heuristics. Thus, the optimal parameter combinations might differ for the different job arrival environments.

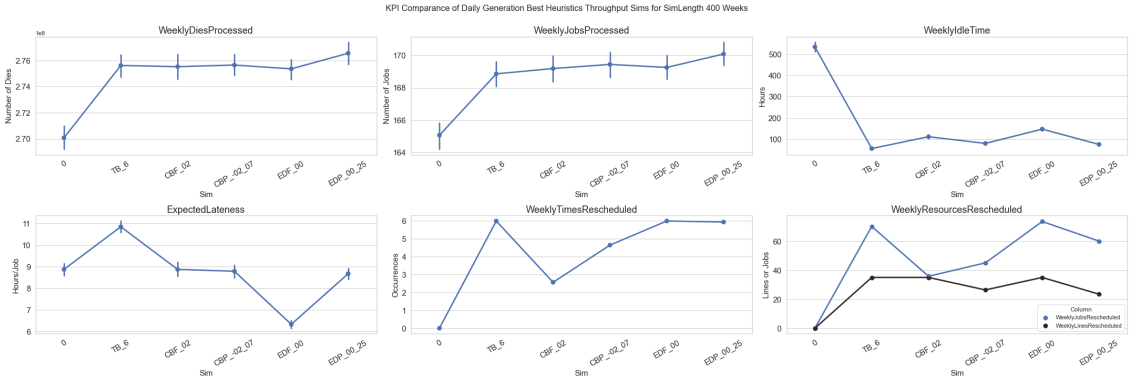


Figure 7.14: Daily generation results with throughput heuristics

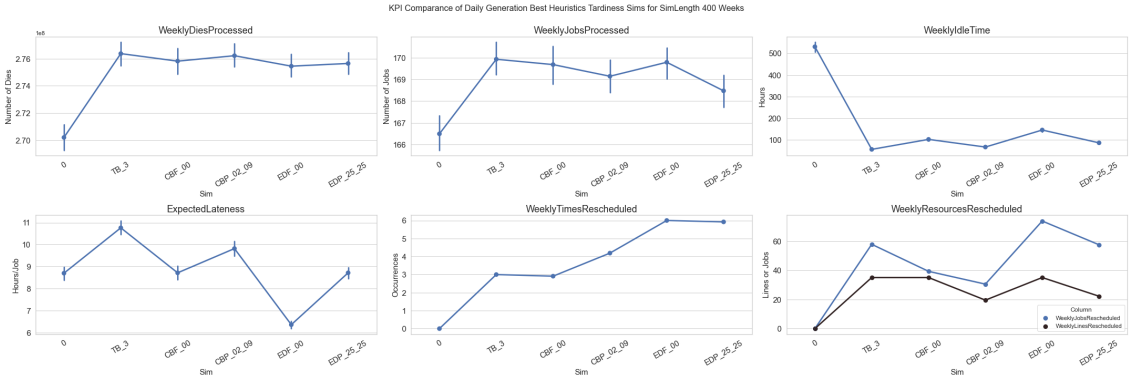


Figure 7.15: Daily generation results with tardiness heuristics

7.6.2 Down Time Included

In Subsection 6.2.2, it is explained how the scheduler determines the predicted production times, without taking down time into consideration. Nevertheless, it is interesting to see what happens to the results when down time is taken into account. In order to do that, the average down ratio over all observations is determined, which is equal to 0.278. When the scheduler determines the predicted process times, the average down ratio is now used to add down time to the predicted production time.

The environment is tested with the *time based* heuristic and the results are shown in Figure 7.16. The heuristic performance behaviour is almost identical to Figure 7.7. However, the KPI values did decrease slightly. The baseline throughput performance is decreased with 0.15% and the tardiness performance is decreased with 2.34%, due to the implementation of down time. The heuristic performances also declined, with the throughput decreasing around 0.20% and the tardiness performance decreases with 4.1%. It can thus be concluded that it is better for the performance to schedule without taking the average down time into account.

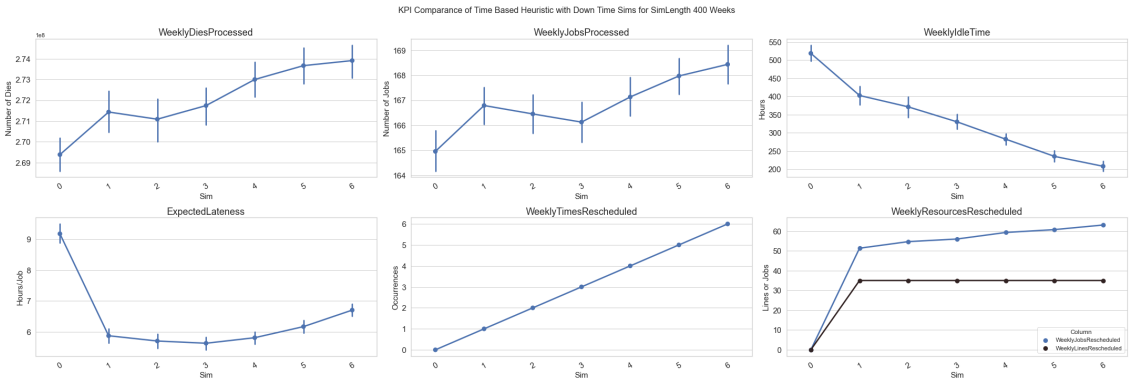


Figure 7.16: Time based heuristic with down time

Chapter 8

Conclusions & Recommendations

8.1 Conclusions

This research aimed to create an autonomous batch scheduler system by creating a closed-loop control system. Based on this control system, the batch scheduler should be able to determine how and when to reschedule. It can be concluded, that the developed system is able to reschedule, improving the performances significantly.

Five sub-research questions were formed at the start of this research. The first question aimed to test the reliability of the current system. To answer this question, the initial factory plans were to be compared to the resulting data. However, due to the lack of data on the initial plans and schedules and the many errors that appeared in the resulting data, no answers could be derived. However, based on the tracking software created and the resulting database, a live tracker has been developed, which can be used by the factory to evaluate their current process.

The second question aims to determine how to implement a rescheduling approach. Within the simulation, four variables are required to implement the different rescheduling heuristics. These variables are the current time, the predicted workload, the scheduled end times and the number of active lines. All these variables are available in the real world and thus it can be concluded that all derived heuristics can be implemented.

The third question considers the threshold values of the different heuristics. The heuristics are either time-based or event-based, which determines how the heuristic is triggered. For each heuristic, a minimal set of parameters is derived to determine when to reschedule. All these parameters are available in the real-world system. In Section 6.4, it is exactly explained how the heuristics are triggered.

The fourth question aims to prevent the need to reschedule. Preventing the need to reschedule requires creating a more robust schedule. In order to that, more knowledge about unexpected events should be taken into account. This is tested with the average down time taken into account with the predicted production times. The resulting performances only decreased, even without rescheduling. For the scope of this research, it can be concluded that it is better to reschedule than preventing the need to reschedule.

The last question is about how to reschedule. Both partial and fully rescheduling heuristics have been tested and their performances have been compared. Based on the results of this research, it can be concluded that fully rescheduling performs better with respect to the throughput, whereas partial rescheduling performs better with respect to the tardiness performance. When comparing the *capacity balance* heuristic to the *end time drift* heuristic, the main difference is that the *capacity balance* reflects the system performance with respect to the end of the schedule. The *end time drift* heuristic evaluates the performance based on the current moment in time. It appears that *capacity balance* performs better for throughput optimization whereas *end time drift* performs better for tardiness optimization.

Based on this research, it can be concluded that rescheduling improves the system performances

significantly. For throughput optimization, the number of dies processed can be increased with 1.71% per week when using the *time based* rescheduling heuristic. This is with the limitation of rescheduling only once per day taken into account and jobs arriving at the start of the week. Increasing the rescheduling frequency can increase this gain to 2.15%. With the three back-end facilities of Nexperia producing 90 billion products annually, this gain with rescheduling is definitely worth the implementation. For the tardiness optimization, the expected lateness per job can be decreased with 44.70%.

When adapting the simulation job arrival rate, the throughput performance increases even more when rescheduling is used. The tardiness performance seems to degrade with daily job arrival. Furthermore, the parameter combinations used for the weekly job arrivals seem to be less optimal with the daily job arrival.

8.2 Discussion & Recommendation

The final results of the simulation show significant improvements of the system performance. Nonetheless, several choices and assumptions have been made to create the simulation. The simulation uses a weekly job arrival rate, which is determined by the workload of the system, and only produces schedules for the current week. The real-world factory has a higher job arrival rate, without limitation of the current workload in the system. Furthermore, ATSN will probably use a rolling time horizon, which will always create schedules with a length of seven days. This might influence the results and is interesting to investigate for further research. A small part of this research already showed that the gain can be increased when using daily job arrival rates. However, this simulation was not initially designed for daily arrivals, neither are the parameters. It might therefore be interesting to design a simulation which is initially aimed at daily job arrivals with a rolling time horizon.

The data obtained from the server is limited to the past 14 months. This data is critical to the foundation of the simulation. During these 14 months, the factory has been closed twice due to the impact of the corona pandemic. It is explained how the data is gathered, cleaned and transformed. It is assumed that the influence of the closures is minimal due to the high amount of total entries, but it might still influence the data. Furthermore, since the research is so dependent on the available data, it might be that analysing another back-end factory provides different conclusions.

During the implementation of the rescheduling heuristic, several limitations have been tested. One of those limitations is that rescheduling can only occur once per day, since this is required for practical implementation. For academic research, it might be interesting to see the influence of rescheduling more often, since the *time based* heuristic showed improving results for a higher rescheduling frequency.

Bibliography

[Adan et al., 2018] Adan, J., Akcay, A., Stokkermans, J., and Dobbelsteen, R. V. d. (2018). A Hybrid Genetic Algorithm for Parallel Machine Scheduling at Semiconductor Back-end Production. *Association for the Advancement of Artificial Intelligence*. 3, 28

[Adan et al., 2020] Adan, J., Deenen, P., and Geurtsen, M. (2020). CSSL Simulation Library. 20, 25

[Bidot et al., 2008] Bidot, J., Vidal, T., Laborie, P., and Beck, J. (2008). A theoretic and practical framework for scheduling in a stochastic environment. *Springer Science+Business Media, LLC*. 7, 9

[Geurtsen and Adan, 2020] Geurtsen, M. and Adan, J. (2020). Integrated Maintenance and Production Scheduling. *Department of Industrial Engineering, Eindhoven, University of Technology*. 1

[Hitachi, 2020] Hitachi, H.-T. C. (2020). History of semiconductors. 1

[Holloway et al., 1977] Holloway, C. A., Nelson, R. T., and Mei-Lun Wong, R. (1977). Centralized Scheduling and Priority Implementation Heuristics for a Dynamic Job Shop Model. *Standford University, A I I E Transactions, 9:1, 95-102, DOI: 10.1080/05695557708975127*. 7

[Insights, 2020] Insights, F. B. (2020). Semiconductor Market Worth. 1

[Kizilisik, 2001] Kizilisik, B. (2001). Analysis of scheduling problems in dynamic and stochastic FMS environment. *Institute of Engineering and Sciences*. 10

[Liu et al., 1997] Liu, T.-H., Trappey, A. J., and Fu-Wei, C. (1997). A Scheduling System for IC Packaging Industry Using STEP Enabling Technology. *IEEE Transactins on Components, Packing, and Manufacturing Technology*, 20:4. 10

[Nexperia, 2020] Nexperia (2020). Nexperia Fact Sheet. 2

[Pearn et al., 2007] Pearn, W., Chung, S., and Lai, C. (2007). Scheduling Integrated Circuit Assembly Operations on Die Bonder. *IEEE Transactins on Electronoics Packing Manufacturing*, 30:2. 10, 13

[Sabuncuoglu and Kizilisik, 2010] Sabuncuoglu, I. and Kizilisik, O. B. (2010). Reactive scheduling in a dynamic and stochastic FMS environment. *International Journal of Production Research*, 41:17:4211–4231. 7

[Statista, 2020] Statista (2020). Global IoT and non-IoT connections 2010-2025. 1

[Tormos and Lova, 2003] Tormos, P. and Lova, A. (2003). An efficient multi-pass heuristic for project scheduling with constrained resources. *International Journal of Production Research*, 41:1071–1086. 7

[Vermunt, 2021] Vermunt, J. C. (2021). Combining machine learning and genetic algorithms: A study on the back-end production process in the semiconductor industry. *Artificial Intelligence and Operations Research, Jheronimus Academy of Data Science*. 21, 22, 29

[Vieira et al., 2000] Vieira, G. E., Herrmann, J. W., and Lin, E. (2000). Predicting the Performance of Rescheduling Strategies for Parallel Machine Systems. *Journal of Manufacturing Systems*, 19:4:256 – 266. 10

[Vieira et al., 2003] Vieira, G. E., Herrmann, J. W., and Lin, E. (2003). Rescheduling Manufacturing Systems: A Framework of strategies, policies and methods. *Journal of Scheduling*, 6:39 – 62. 10

[Wang and Jiang, 2015] Wang, C. and Jiang, P. (2015). Manifold learning based rescheduling decision mechanism for recessive disturbances in RFID-driven job shops. *Springer Science+Business Media, NY*. 10

Appendix A

Validations

A.1 Simulation Validation Model

The simulation required the implementation of single-server and multi-server systems. A schematic overview of the multi-server system is shown in Figure A.1. The system starts with a job generator, which creates job instances using an exponential distribution of λ for the interval rate, and queues the jobs into the system. A dispatcher is assigned to this queue and dispatches the jobs over the different machines. The machines process the jobs with an exponential distribution of μ. All jobs that are finished are logged by the factory observer.

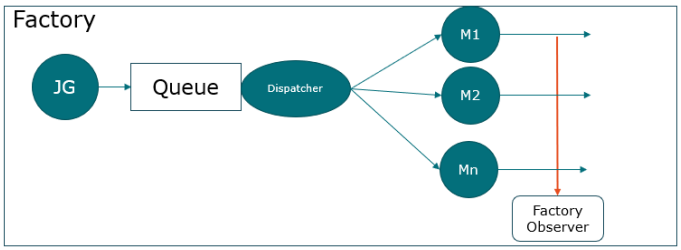


Figure A.1: Multi-server system schematics

The results are validated based on three parameters. First of all, $E[N]$, the average number of jobs in the system. Secondly, $E[T]$, the average time a job is in the system. Finally, $E[T_q]$, the average time a job is in the queue. Both systems are simulated for a smaller and larger number of jobs and for different combinations of λ and μ. 30 replications are performed and combined to minimize the influence of noise. All combinations are solved by hand and the results are compared to the results of the simulations.

For the single-server system, the parameters can be solved using the following equations:

$$\rho = \frac{\lambda}{\mu} \tag{A.1}$$

$$E[N] = \frac{\rho}{1-\rho} \tag{A.2}$$

$$E[T] = \frac{1}{\mu - \lambda} \tag{A.3}$$

$$E[T_q] = \frac{\rho}{\mu - \lambda} \tag{A.4}$$

The system is validated for two different combinations of λ and μ. The simulation ran 30 replications and the average result is shown in Table A.1. It can be seen that the shorter simulation has some variation to the deterministic approach but the longer simulation is almost identical to

the results derived by hand. Thus is can be concluded that the single-server $M/M/1$ system is correctly implemented.

Table A.1: Validation results M/M/1

	Check 1 Hand	Check 1 C# [250 jobs]	Check 1 C# [25k jobs]	Ratio	Check 2 Hand	Check 2 C# [250 jobs]	Check 2 C# [25k jobs]	Ratio
Lambda	1/4	1/4	1/4	1.000	2/5	2/5	2/5	1.000
Mu	1/3	1/3	1/3	1.000	2/3	2/3	2/3	1.000
E[N]	3	2.6950	2.9919	0.8983 0.9973	1.5	1.5053	1.5005	1.0035 1.0003
E[T]	12	10.6578	11.9719	0.8882 0.9977	3.75	3.7860	3.7500	1.0096 1.0000
E[T_q]	9	7.7171	8.9730	0.8575 0.9970	2.25	2.2825	2.2503	1.0144 1.0000

For the multi-server system, the following equations are used to solve the parameters, where s is the number of servers used:

$$\rho = \frac{\lambda}{\mu s} \tag{A.5}$$

$$P_0 = [\sum_{n=0}^{s-1} \frac{1}{n!}(\frac{\lambda}{\mu})^n + \frac{1}{s!}(\frac{\lambda}{\mu})^s \frac{s\mu}{s\mu - \lambda}]^{-1} \tag{A.6}$$

$$E[N_q] = [\frac{1}{(s-1)!}(\frac{\lambda}{\mu})^s \frac{\lambda\mu}{(s\mu - \lambda)^2}]P_0 \tag{A.7}$$

$$E[N] = E[N_q] + \frac{\lambda}{\mu} \tag{A.8}$$

$$E[T] = \frac{E[N_q]}{\lambda} + \frac{1}{\mu} \tag{A.9}$$

$$E[T_q] = \frac{E[N_q]}{\lambda} \tag{A.10}$$

The system is validated for two different combinations of λ and μ and s. The results are shown in Table A.2. The table shows that results for the shorter replication are promising but not feasible. For the longer simulation, the results are almost identical compared to the results by hand. It can be concluded that the $M/M/s$ system is correctly implemented.

Table A.2: Validation results M/M/s

	Check 1 Hand	Check 1 C# [250 jobs]	Check 1 C# [25k jobs]	Ratio	Check 2 Hand	Check 2 C# [250 jobs]	Check 2 C# [25k jobs]	Ratio
Lambda	1/2	1/2	1/2	1.000	2/5	2/5	2/5	1.000
Mu	1/3	1/3	1/3	1.000	1/4	1/4	1/4	1.000
s	3	3	3	1.000	5	5	5	1.000
E[N]	1.7368	1.7388	1.7386	1.0012 1.0010	1.6122	1.6214	1.6127	1.0057 1.0003
E[T]	3.4737	3.4487	3.4752	0.9928 1.0004	4.0304	4.0673	4.0296	1.0092 0.9998
E[T_q]	0.4737	0.4769	0.4748	1.0068 1.0023	0.0304	0.0322	0.0307	1.0592 1.0099

A.2 Speed Validation

In Section 7.1, a brief validation of the production speed and the effective production speed is presented. However, just comparing the figures does not provide enough insights. This section will provide extra details to confirm the correct implementation.

As explained before, twenty subpopulations are defined, based on the consume factor and the die bonder model. For each die bonder model, the data is shown in the tables below. On the horizontal lines, the simulation data and historical data are shown as well as the ratio. The ratio shows the simulated average divided by the historical average. If the ratio is below 1, the historical data has higher production speeds. If the ratio is above 1, the simulation data has higher production speeds. In the columns, each consume factor shows the number of observations taken into account, the average production speed (PS) over those observations and the average effective production speed (EPS).

The first model is the *ADAT 2 14k*, which can only produce products with consume factor 1 or 2. It can be seen in Table A.3 that there are only a few observations for consume factor 1 but many observations for consume factor 2. Due to the low number of observations, the ratio for consume factor 1 is poor but the ratio for consume factor 2 is really good. Thus, with enough observations, the real-world behaviour is correctly mimicked.

Table A.3: Speed validation ADAT 2 14K

	CF_1 obs	CF_1 PS	CF_1 EPS	CF_2 obs	CF_2 PS	CF_2 EPS	CF_3 obs	CF_3 PS	CF_3 EPS	CF_4 obs	CF_4 PS	CF_4 EPS
Simulation data	22	14718	8118	1740	8306	5751	x	x	x	x	x	x
Historical data	5	14819	10225	463	8325	5723	x	x	x	x	x	x
Ratio		0.993	0.794		0.998	1.004	x	x	x	x	x	x

The second model is the *ADAT 2 18K*, with similar limitations as the *ADAT 2 14K*. The results are shown in Table A.4. The observation count is really high for this model and the results are very good. With an accuracy of $96 - 100\%$, the real-world behaviour is correctly mimicked.

Table A.4: Speed validation ADAT 2 18K

	CF_1 obs	CF_1 PS	CF_1 EPS	CF_2 obs	CF_2 PS	CF_2 EPS	CF_3 obs	CF_3 PS	CF_3 EPS	CF_4 obs	CF_4 PS	CF_4 EPS
Simulation data	1198	17189	10230	35593	8621	6192	x	x	x	x	x	x
Historical data	56	17188	10920	10024	8622	6463	x	x	x	x	x	x
Ratio		1.000	0.937		0.999	0.958	x	x	x	x	x	x

The third model is the *ADAT 3 36K*, which is capable of producing all consume factors. It is noted that the production speed is really accurate for most subpopulations, only with consume factor 4 there is a small deviation. With the effective production speed, the results for consume factor 1 and 3 are relatively low. The results for consume factor 2 are really good, which has a significant higher observation count than the other subpopulations. It can be concluded that the production speed is almost perfectly implemented and the effective production speed is good, with an increasing accuracy when the observation count increases.

Table A.5: Speed validation ADAT 3 36K

	CF_1 obs	CF_1 PS	CF_1 EPS	CF_2 obs	CF_2 PS	CF_2 EPS	CF_3 obs	CF_3 PS	CF_3 EPS	CF_4 obs	CF_4 PS	CF_4 EPS
Simulation data	341	29075	10380	7200	16860	11176	164	11707	7760	324	8784	5397
Historical data	51	29369	11896	988	16856	11114	105	11694	9639	37	9215	5832
Ratio		0.990	0.873		1.000	1.005		1.001	0.805		0.953	0.925

The fourth model is the *ADAT 3 48K*, also capable of producing all consume factors. The first thing noted, is the high production speeds compared to the other tables. Except for consume factor 1, which has only a few observations, all production speeds are mimicking the real-world

behaviour with almost 100% accuracy. The effective production speed is slightly less accurate, but still within 10% deviation.

Table A.6: Speed validation ADAT 3 48K

	CF_1 obs	CF_1 PS	CF_1 EPS	CF_2 obs	CF_2 PS	CF_2 EPS	CF_3 obs	CF_3 PS	CF_3 EPS	CF_4 obs	CF_4 PS	CF_4 EPS
Simulation data	30	31361	17221	14870	19347	13544	2838	14254	10115	1372	10632	7507
Historical data	46	33375	16091	3330	19345	12840	422	14258	11218	237	10640	8042
Ratio		0.940	1.070		1.000	1.055		0.999	0.901		0.999	0.934

The last model is the *ADAT 3 Glue*, which is only capable of producing devices with consume factor 3 as shown in Table A.7. There are only 37 observations in the historical data. It can be seen that production speed is accurately implemented. The effective production speed has a poor performance compared to the other ratios, with the simulation producing slower than the actual factory.

Table A.7: Speed validation ADAT 3 Glue

	CF_1 obs	CF_1 PS	CF_1 EPS	CF_2 obs	CF_2 PS	CF_2 EPS	CF_3 obs	CF_3 PS	CF_3 EPS	CF_4 obs	CF_4 PS	CF_4 EPS
Simulation data	x	x	x	x	x	x	647	10199	7237	x	x	x
Historical data	x	x	x	x	x	x	37	10239	9278	x	x	x
Ratio	x	x	x	x	x	x		0.996	0.78	x	x	x

Based on all the observations, a weighted average ratio is determined for the production speed and for the effective production speed. The weighted average ratio is determined using Equation (A.11).

$$AWR = \sum_{m=1}^{m=M} \sum_{cf=1}^{cf=4} (r_{i,m} \cdot obs_{i,m}) / obs_T \tag{A.11}$$

AWR is the average weighted ratio, with M being the number of different of machine models, in this case five. $r_{i,m}$ is the ratio determined for the combination of i, the consume factor, and m, the machine model. This ratio is multiplied by the observation count. The total sum is divided by obs_T, the total number of observations. It is chosen to use the simulation observations as observation count, which has a total of 66 339 observations. The resulting average weighted ratios are shown in Table A.8.

Table A.8: Average weighted ratio

	Average ratio
Production Speed	0.999
Effective Production Speed	0.980

It can be concluded that the overall performance of the production speed and the effective production speed are really accurate in mimicking the real-world behaviour. On average, the simulated production speeds are slightly lower than the actual production speeds. The implementation of the down time results in the effective production speed being lower than the real-world effective production speed. But this is still within 2% deviation which is a feasible result.

Appendix B

Magnified Figures

Throughout the report, several figures are presented. Due to space constraints on the pages, some figures are small in size and might not be easy to read. This appendix will provide all figures considering data in a magnified version.

Data Acquisition

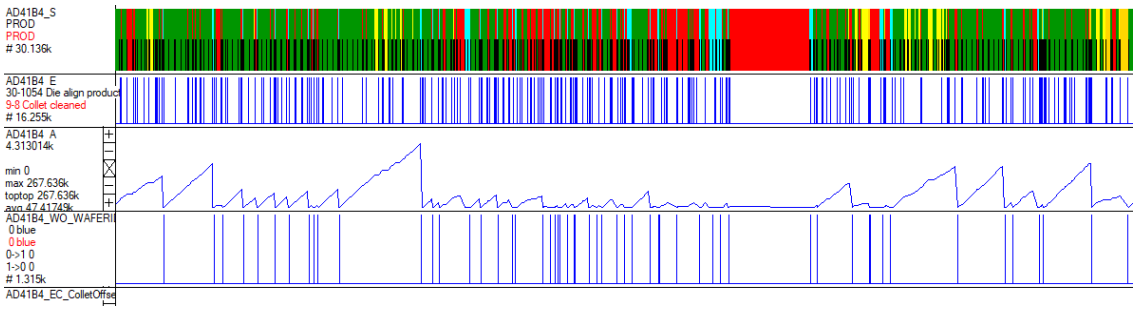


Figure B.1: Magnified version of visual representation of an ESM file

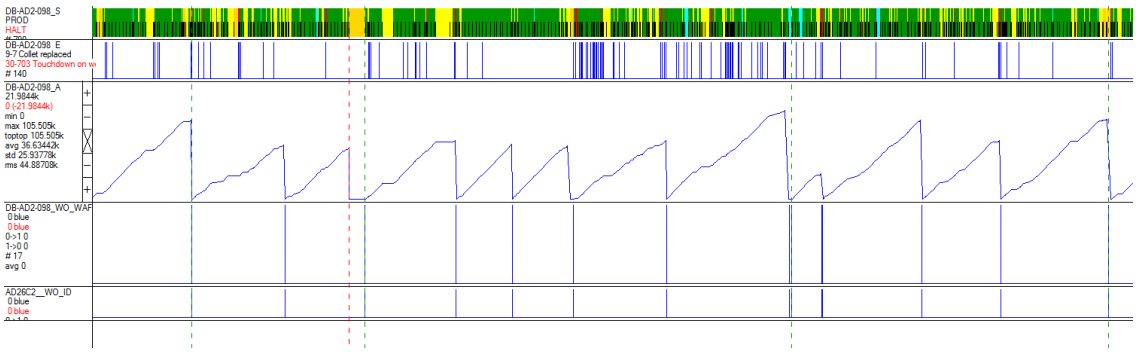


Figure B.2: Magnified version of combined ESM file

Data Exploration

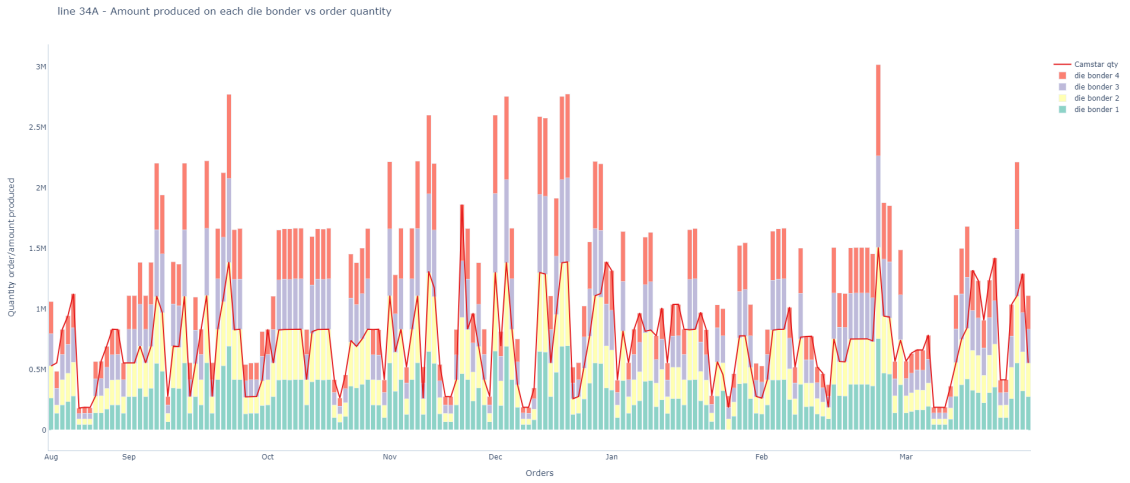


Figure B.3: Magnified version of overproduction due to off Consume Factor

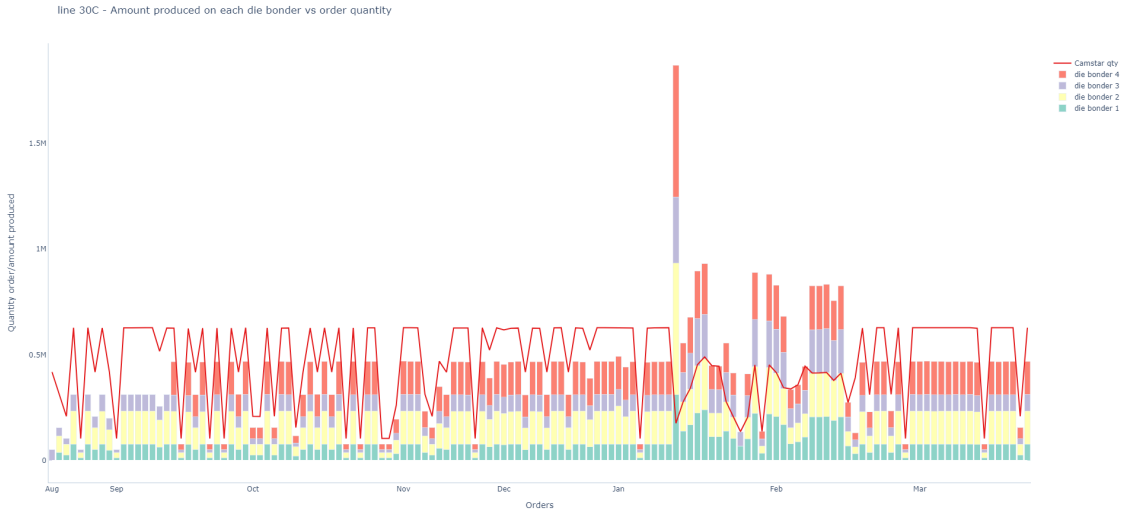


Figure B.4: Magnified version of underproduction due to off Consume Factor

Simulation

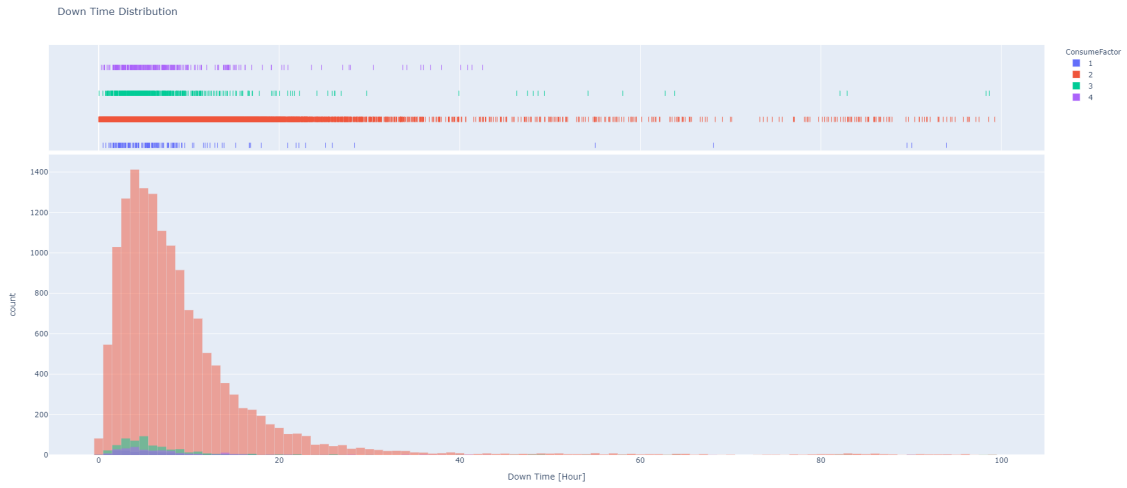


Figure B.5: Magnified version of down time distribution based on consume factor

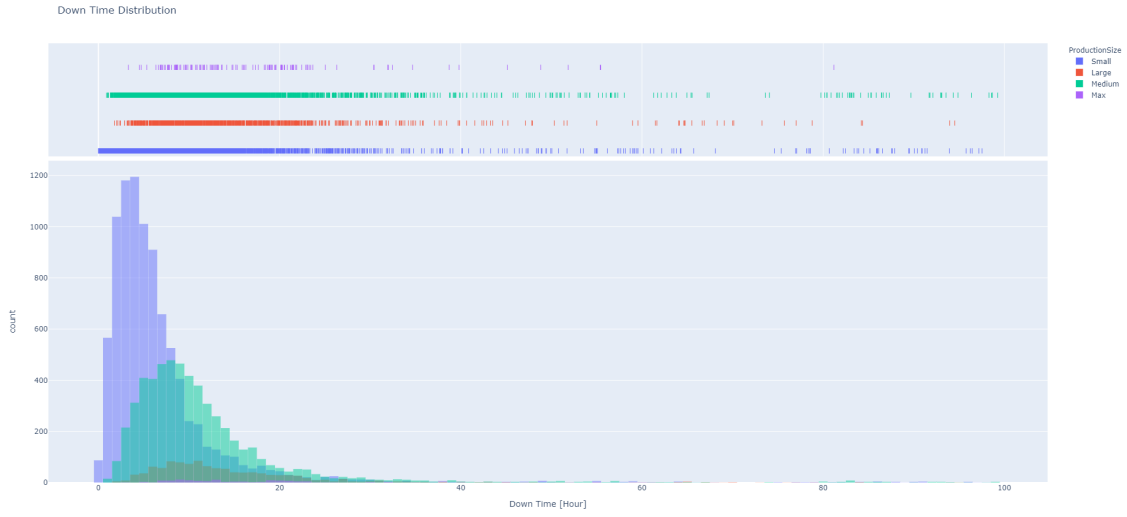


Figure B.6: Magnified version of down time distribution based on order quantity

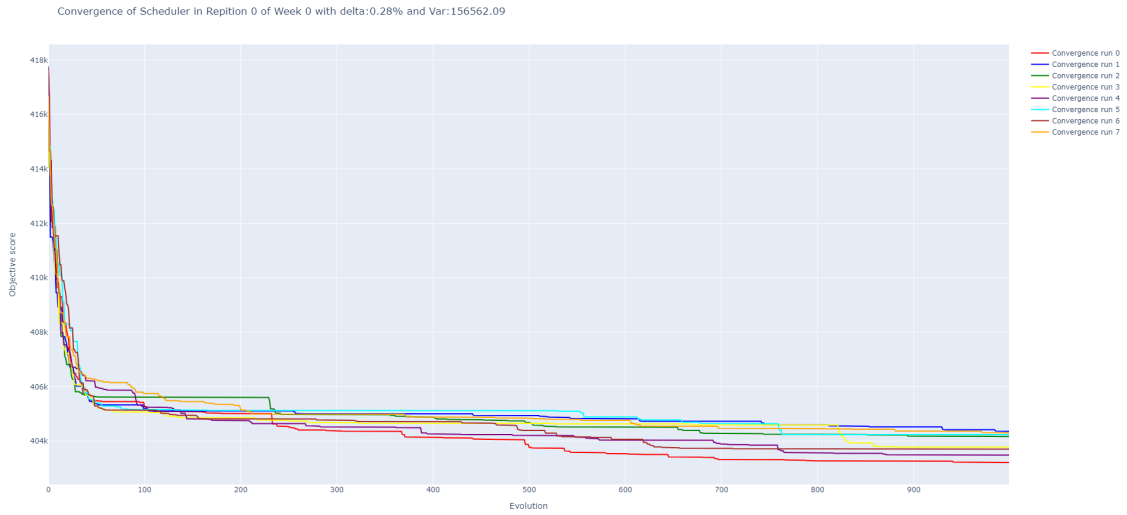


Figure B.7: Magnified version of objective score week 1

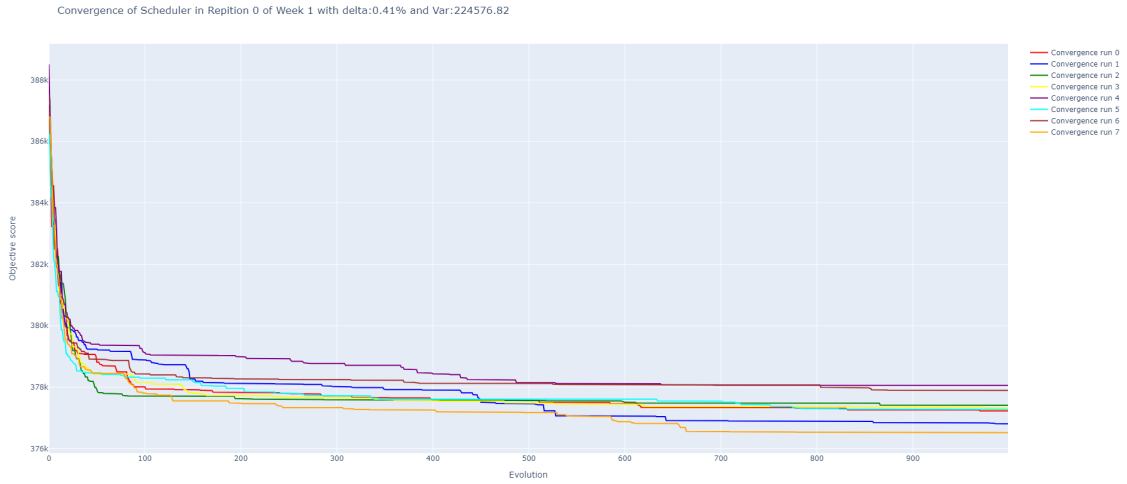


Figure B.8: Magnified version of objective score week 2

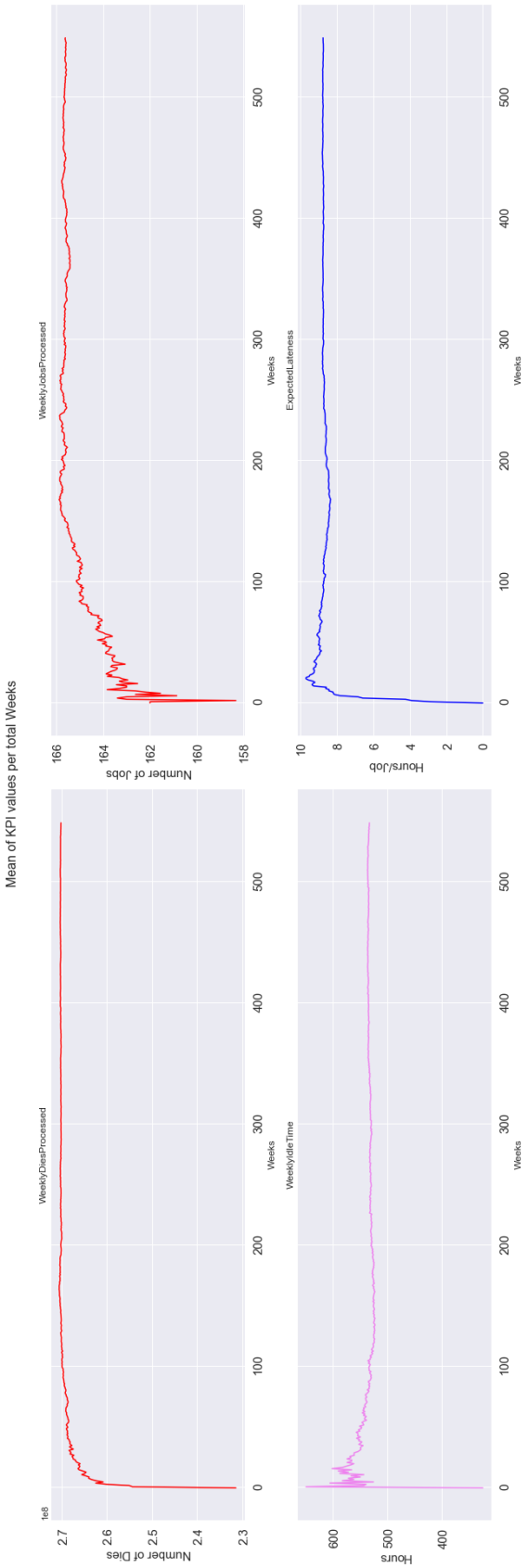


Figure B.9: Magnified version of KPI convergence Mean

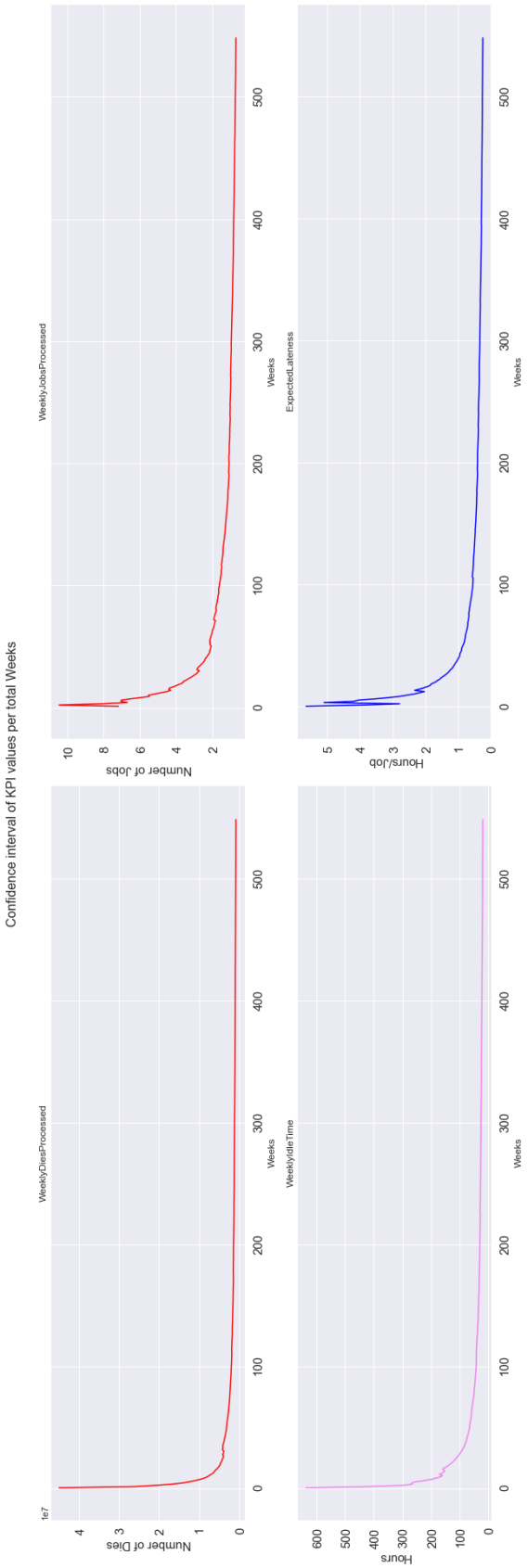


Figure B.10: Magnified version of KPI convergence Confidence Interval

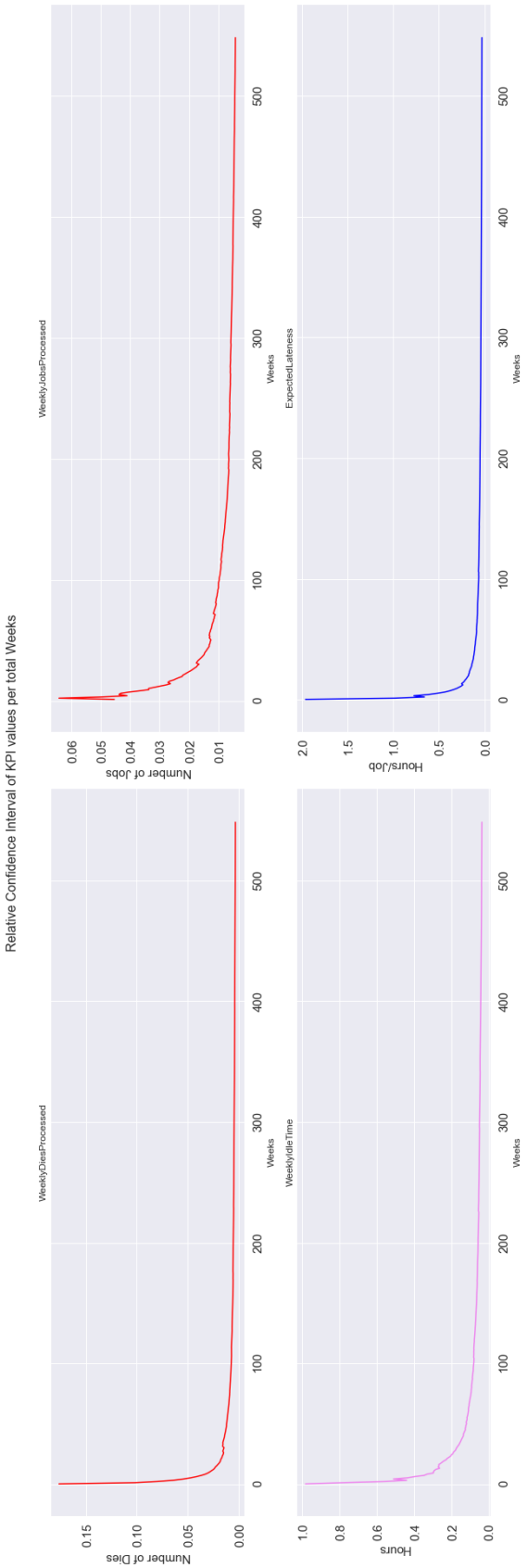


Figure B.11: Magnified version of KPI convergence Relative Confidence Interval

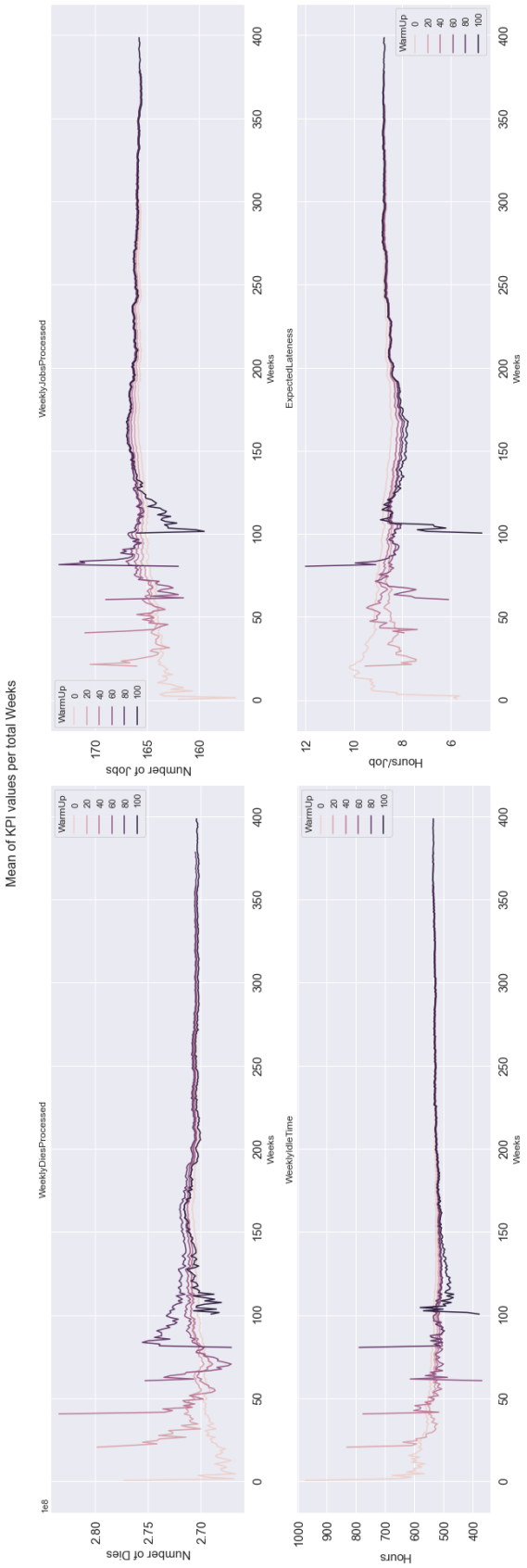


Figure B.12: Magnified version of KPI convergence Mean

Results

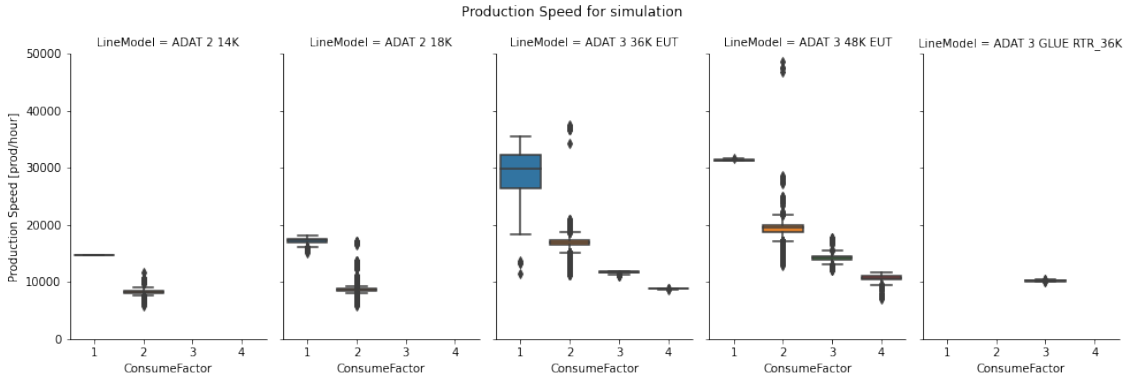


Figure B.13: Magnified version of production speed simulation data

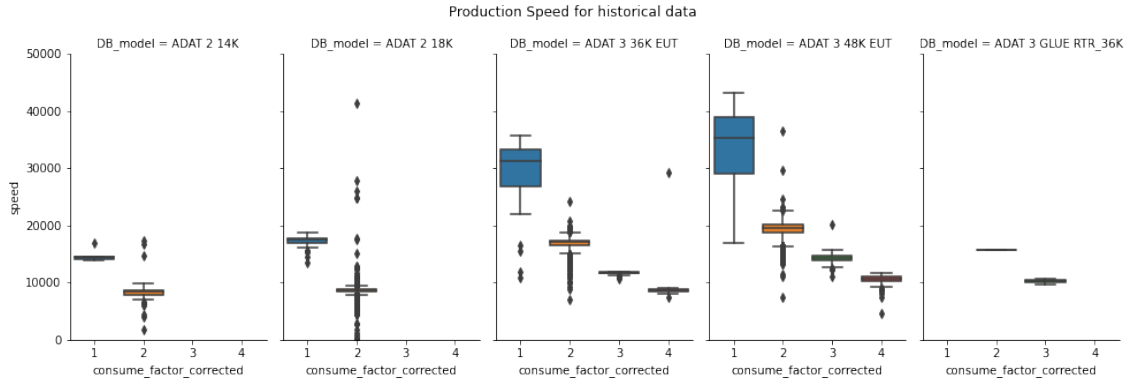


Figure B.14: Magnified version of production speed historical data

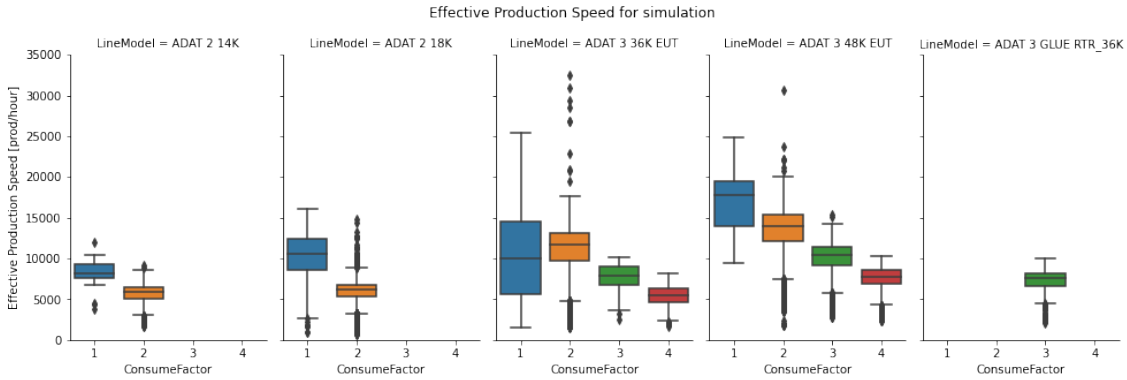


Figure B.15: Magnified version of effective production speed simulation data

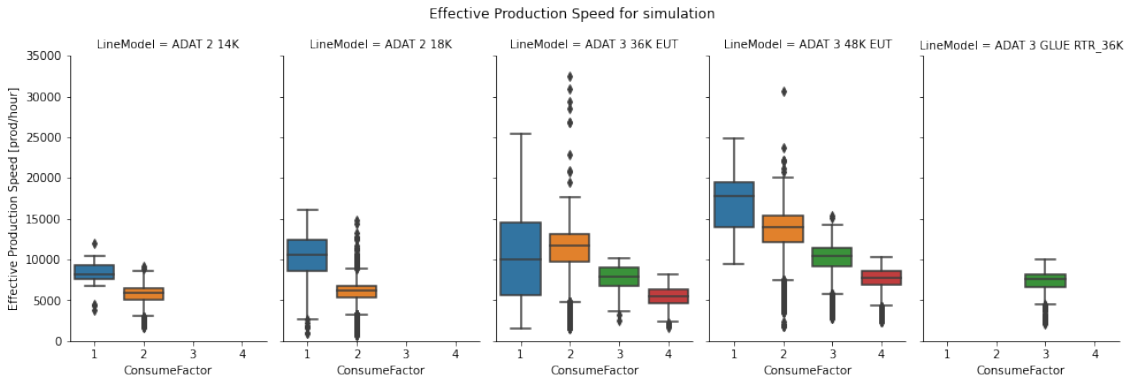


Figure B.16: Magnified version of effective production speed historical data

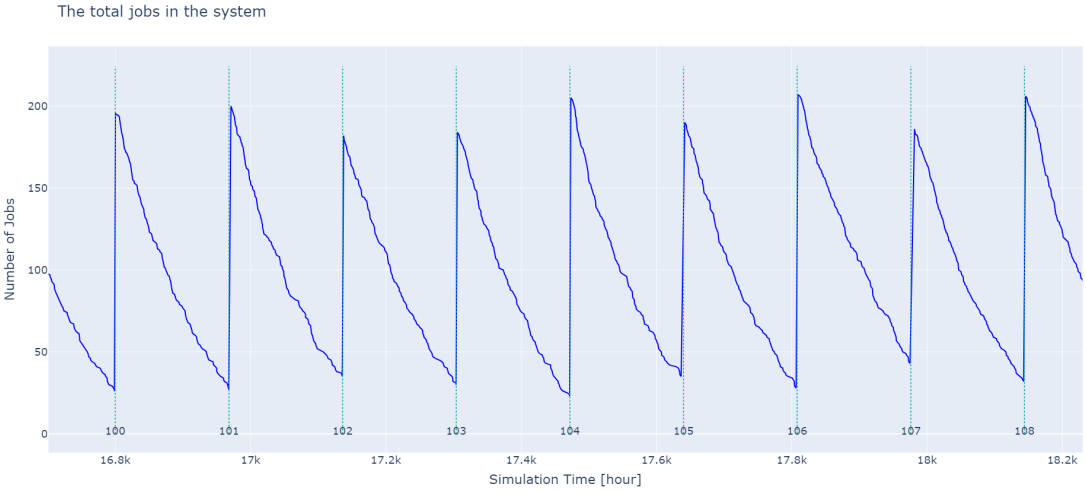


Figure B.17: Magnified version of baseline performance - total jobs in the system

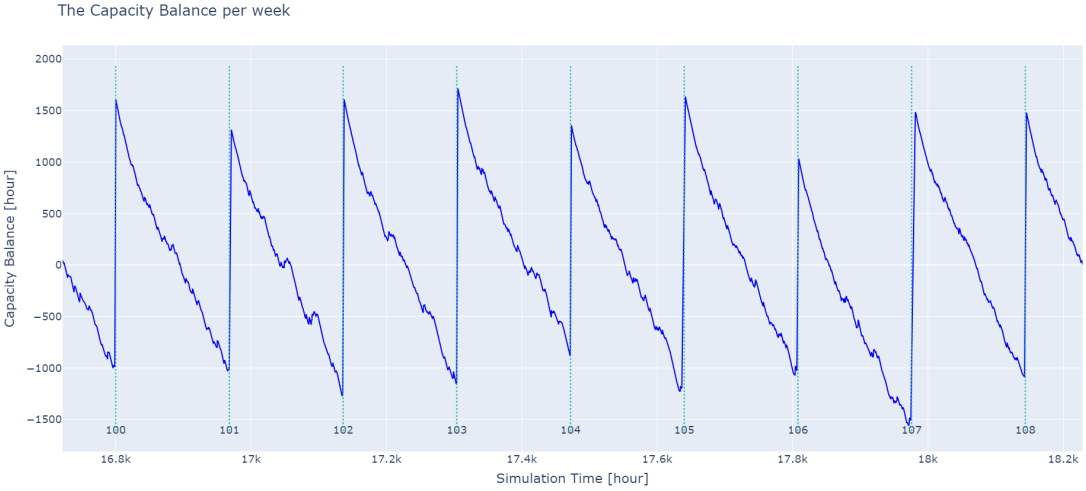


Figure B.18: Magnified version of baseline performance - capacity balance of the system

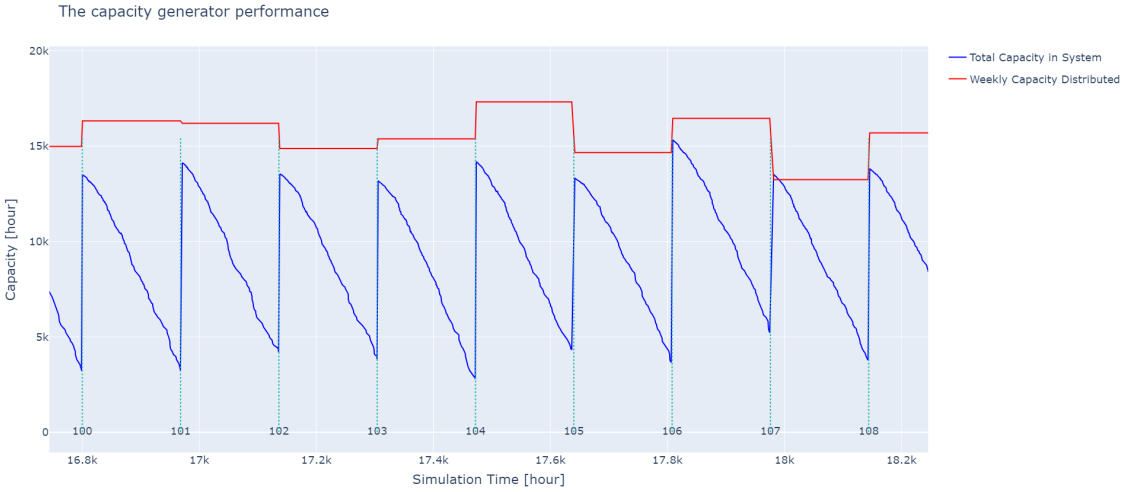


Figure B.19: Magnified version of baseline performance - capacity generator performance

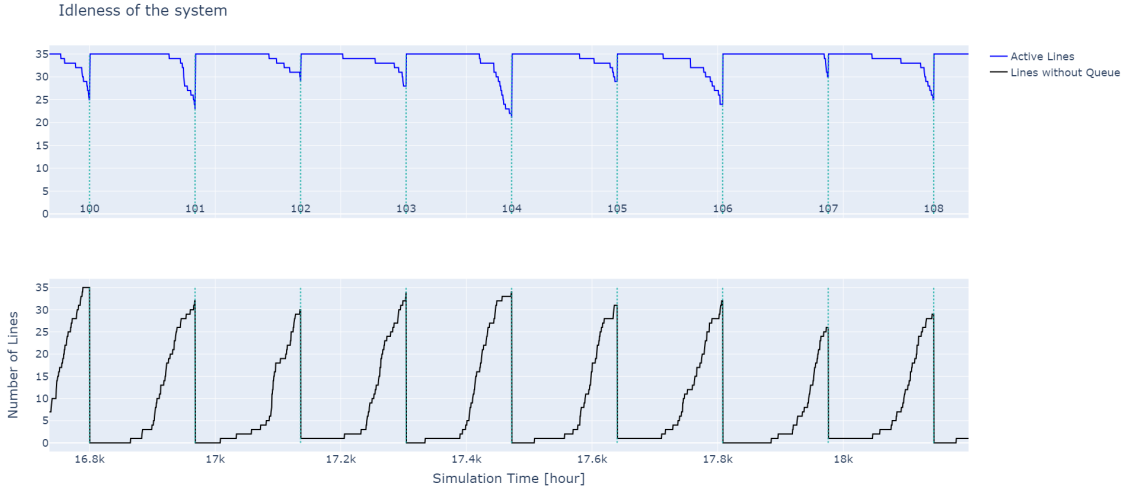


Figure B.20: Magnified version of baseline performance - idleness performance of the system