CODEN:LUTEDX/(TEIE-5458)/1-62(2021)

Industrial Electrical Engineering and Automation

Study of an Automated Testing Procedure with Dynamic Job Scheduling and Automatic Error Handling

Tom Andersson

Division of Industrial Electrical Engineering and Automation
Faculty of Engineering, Lund University

Study of an automated testing procedure with dynamic job scheduling and automatic error handling

Tom Andersson
tomandersson94@gmail.com

April 14, 2021

Abstract

During the production of Axis's cameras, the optics modules have to do a number of tests and calibrations that are performed in different test stations. The test and calibration sequence differs between different models. This has so far been done manually by having operators move the units between test stations. However, Axis are planning to automate the procedure by having an industrial robot move test units between the different test stations. The scope of this thesis is to come up with a concept for the control of the system, including the flow of information and the job scheduling, and implement a simulation program where production sequences can be evaluated. The thesis is divided into four parts, specifying the system requirements, concept generation, scheduling analysis, and implementation of simulations.

The requirements of the system were formulated by discussing the vision of the system with the project team at Axis. The essential requirements can be summarized as follows. The throughput of the system and the utilization of the different test stations should be as high as possible. It should be possible to connect and disconnect test stations during production, without interrupting the rest of the system. The system should be able to automatically handle unexpected states, such as test units being moved, having units already within the system upon start up, test stations which break during production etc. The system should handle potential deadlock situations to keep the production running. And finally, relevant data concerning tests, test stations and the flow of production should be stored for traceability reasons and visualized to achieve an overview of the production.

During the concept generation two concepts were generated. One concept was based on the initial visions of the project team at Axis. The fundamental idea with this concept is that information which is both generated within the system and is needed further down the test sequence of a specific test unit is stored on an ID-tag, for example an RFID-tag, attached to that test unit. This information is thereafter read when the unit arrives to the different subsystems within the system. The other concept was generated more freely based on the knowledge and experience of the student carrying out the thesis. In this concept, the generated information is instead stored in a database and thereafter accessed by the different subsystems within the system. It was concluded that both concepts offer similar functionalities but the complexity of the ID-tag concept was lower. However, the project team at Axis was more experienced in working with databases compared to ID-tags and it was assessed that the lower complexity of the ID-tag concept was not significant enough to overlook the experience of the team. Therefore, the database concept was recommended for a future implementation.

To achieve an efficient system, the theories of job shop scheduling are analysed. A framework for dynamic scheduling was studied to get an overview of how job scheduling problems can be solved. For instance, the framework described three different categories of dynamic scheduling approaches: heuristic rules, classical optimization and approaches based on artificial intelligence. Within these categories, a couple of major techniques were analysed namely 12 standardized heuristic rules for different objectives, the classical optimization techniques dynamic programming and branch and bound, A*, as well as the artificial intelligence approaches beam search, genetic algorithms, and tabu search. One technique from each category was also analysed on a deeper level with focus on the system concerned by this theses. These techniques were a combined heuristic rule focusing on throughput, A*, and beam search, where the last technique, beam search, is a combination of heuristic rules and A*.

The implementation of the simulation was based one the database concept. The user specifies the available test stations, occurrence of breakdowns, buffer size, an incoming rate of test units with a given test cycle, test units already placed inside the system upon start up, etc. The simulation program will then run these sequences using the derived combined heuristic rule as well as specified time durations for tests and the movement of the robot. Once the simulations have completed, the user is presented with relevant data such as total time, throughput, utilization factors etc. Optimally the simulations should also include A* and beam search but this was not included in the simulations within the scope of the thesis.

This thesis provides a foundation for a future implementation of the system. The derived system concept can be used as a base for a future system design and the simulation software can be used to evaluate several production capabilities such as, how many test stations of each type that should be used at a certain production rate.

Preface

Years of studies have come to an end and all the knowledge and experience that have been acquired throughout the years are now applied in this thesis. I would like to thank everyone at Axis for giving me this opportunity, for supporting me throughout the thesis, and most of all, for making me feel welcome to the team at Axis. A special thanks to my supervisors Björn Hansson and Martin Nyman, my academic supervisor Gunnar Lindstedt, and my examiner Ulf Jeppsson for all the valuable guidance you have given me during this project. Thank you, it would not have been possible without you.

Contents

Acronyms and abbreviations

com. Communication. 20

DBMS Database Management System. 19

HMI Human Machine Interface. 17, 19

I/O Input/Output. 18

IR Infrared. 18

MCU Microcontroller Unit. 19

PLC Programmable Logic Controller. 7, 10, 17–20

pos. Position. 19

QR Quick Response. 17, 19, 20

RFID Radio Frequency Identification. iii, 17, 19, 20

RW Read-Write. 19

TOML Tom's Obvious Minimal Language. 34, 41

WORM Write Once, Read Many. 19

Chapter 1

Introduction

Background

Among other products Axis Communications develops, produces and sells network cameras [2]. In the manufacturing of the optics modules a number of tests and calibrations are performed on each unit. Which tests and calibrations that are performed and in which order they are performed depends on the type of objective, image sensor and other components. The size of Axis's product portfolio combined with the limited product life cycle makes this a constantly varying procedure. So far the procedure has been carried out by manually moving the optics modules between different testing and calibration machines.

Axis plans to automate this process with the use of an industrial robot. The idea is that when an optics module has been assembled and is ready for testing and calibration, an operator will place it on a small pallet and connect it to a battery and a microprocessor. The unit is then placed at an input station from where an industrial robot takes the unit and moves it to the different testing and calibration stations until the unit is ready and finally placed at an output station. Thereafter, an operator disconnects the optics module from the pallet. To increase the efficiency, several pallets will be in the system at the same time. This is visualized in figure 1.1.

This system will be the core of Axis's future testing procedure of optics modules. It will therefore be placed at the different production factories hired by Axis around the world and it will affect most of the products sold by Axis.

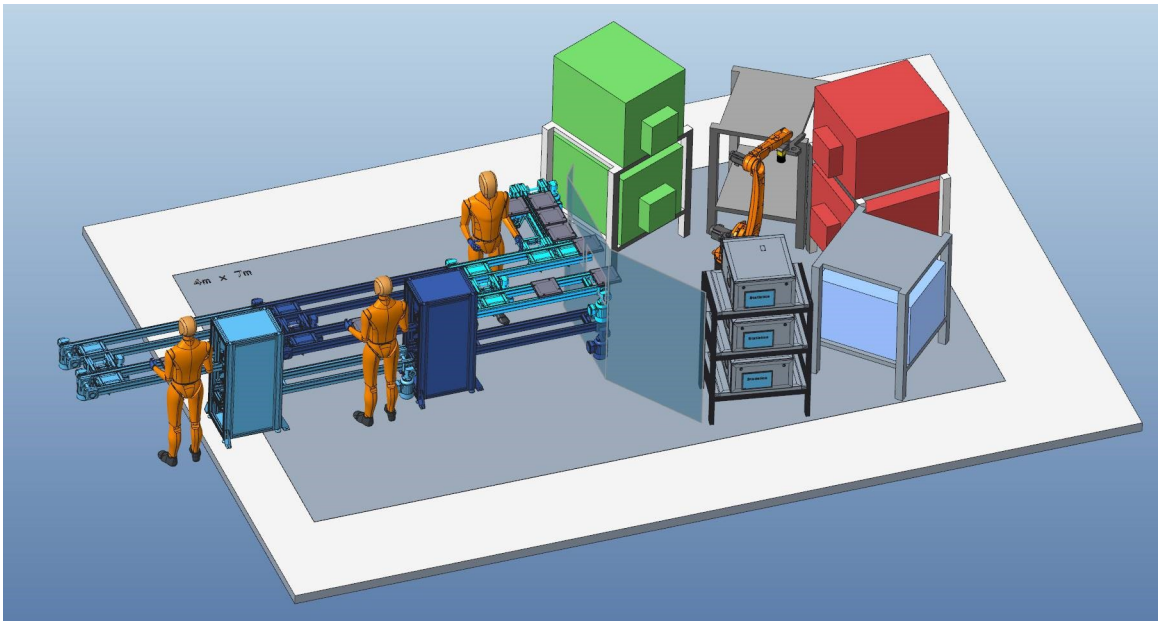

Figure 1.1: An overview of the system. The figure was provided through Axis's internal material.

Goal

The goal of the thesis is to develop a principle for the control of the system including the flow of information required for the robot to achieve its tasks. The resulting principle should enable the following:

- A flexible setup where it is possible to locally, at a specific factory, add and take away testing stations without any need for reprogramming.
- Upon restart after a power failure, the system should automatically determine its state, in terms of the optics modules within the system, and resume production by redoing test cycles that were interrupted by the power failure.
- An automatic handling of tests that have become invalid due to a test station being opened by an operator in the middle of an ongoing test.
- A stable production with a maximized uptime.
- A production where the test stations are being utilized to the maximum.
- Service and replacement of one or several test stations without the other test stations experiencing any downtime, i.e. the system shall remain running with the capacity of the remaining test stations.

Approach

To achieve the goal, this project is approached in the following way:

- Derive at least two different concepts for the principle for controlling the system and describe the flow of information in detail.
- Specify what subsystems, hardware and software, the different principles require.
- Derive the advantages and disadvantages for each of the principles.
- Derive at least one technique for handling the necessary job shop scheduling.
- Simulate production sequences based on at least one of the principles.
- Recommend one of the principles.

Limitations

The project will not include a physical implementation of the system but rather a theoretical study and simulations. Deriving how many test stations of each kind that would be sufficient at a certain production rate and assumed failure rate will not be included within the scope of the project either.

Chapter 2

System specifications

By talking with the employees that are working on the project and the existing testing systems the following specifications were derived.

2.1 Fundamental functionality

An operator will attach the optics module to a fixing plate and the fixing plate to a pallet holding the necessary components to drive the optics module. The fixing plate will be product specific but there will only be one type of pallet. One or potentially several pretests will be performed. If the pretests are successful, the test unit will be placed at one out of two positions in an arrival station using a conveyor. If not, the test unit will either be repaired directly or it will be moved to an area for failed test units.

If the pretests were successful, an industrial robot will take the test unit and move it between different test stations. Within the system, there can be one or several test stations of each type. Different types of optics modules should go through different testing cycles, i.e. different types of test stations, where the order can be either specified, irrelevant or partly specified. In some cases, parameters derived during a test in one test station is needed during a test in a test station further down the test cycle. In some cases, the test cycle should be aborted after a failed test and in some cases the test cycle should continue regardless, depending on the type of optics module and in which test station the failure occurred. The time duration of each test is within a magnitude of seconds or minutes. Each optics module should undertake around five to ten tests.

Axis's main database system will provide test definitions that describe the possible test cycles for each type of optics module. In the test stations, the optics modules are hit with light beams from optical collimators. An optical collimator is a device that creates parallel beams of light from a point source [1]. Different types of optics modules should have the collimators placed at different angles. The main database system will provide the information that tells which angels that should be used for each type of optics module. The results from each test will be stored in Axis's main database system.

Several test units will be going through their test cycles simultaneously by having the robot move one test unit while other test units are being tested. Once a test unit has finished its test cycle or if its test cycle has been aborted, the robot will move the test unit to an output area. The output area can either consist of one area for failed units and one for units that have passed their tests or, the output area can consist of a conveyor that distributes the test units into two different areas depending on if the tests were successful or not. The pallets and fixing plates will be reintroduced to the system and used for new arriving optics modules.

2.2 Efficiency

The utilization of the test stations should be maximized and the average waiting time for the test units should be minimized. It should be possible to set priorities on different types of optics modules and on specific test units, the later for debugging purposes. The estimated time for a test will be estimated beforehand but it should also be reevaluated over time to increase the accuracy of the sorting algorithm. The system will include a buffer area of limited size where test units can be placed in between tests to increase the efficiency.

2.3 Flexibility

It should be possible to add, take away and replace test stations at any time during production and having the scheduler replan accordingly. To keep the dedicated workspace of the robot intact when a test station is moved, the test stations will be structured in a way so that only the interior of the station is being manipulated while the shells of the test stations are being static, forming the workspace of the robot. It should also be possible to at any time perform maintenance on test stations through an opening in the back of the stations while the rest of the system is still active, again having the scheduler replan accordingly. It should be possible to specify new types of optics modules with new types of fixing plates and assign test cycles without pausing the production. It should also be possible to specify new types of test stations that can be included in test cycles without pausing the production. For debugging purposes, it should be possible to edit the test cycle for a specific test unit. Units that performed a different test cycle than the standard test cycle for that type of optics module may not be mixed up with modules that passed their standard test cycles.

2.4 Error handling

To handle unexpected shut downs, for example due to a power failure, the system should when starting up check if there are any modules already in the system, i.e. at the arrival area, the buffer area, in the robot's gripper or in any of the test stations. From that given initial state all unfinished test cycles from the last time the system was powered up should be redone, i.e. all devices that are inside a test station that is not their supposed first test need to be taken back to either their first test station or the buffer area when it is their turn. A module in the robot's gripper needs to either be taken to its first test station or to the buffer area. Which alternative that is chosen and in which order these actions are performed should be derived by the scheduler.

If a test station is opened during a test, the identification of the pallet and fixing point should be reconfirmed and the whole test cycle should be redone. If the components have been changed or if the test unit has been taken out of the test station during the time it was open, the scheduler should replan accordingly. If the test station was opened from the back at the same time as the station is open from the front, which opens when the robot is about to put in a test unit, the workspace of the robot should be considered as entered and the robot should stop and not start again until the back opening has been closed and a start command has been sent manually. If the test station was opened from the back while the station is not open from the front the test station should be considered unavailable and the scheduler should replan accordingly. To eliminate the possibility to tamper with test units in the buffer area unnoticed, this area should be unreachable from outside the robot's workspace. If the entrance to the workspace is opened, not only should the robot stop and not start again until the entrance has been closed and a start command has been sent manually but the test units in the buffer area should also have their identification reconfirmed and their test cycles redone. If a test unit has been taken away or moved to a new position in the buffer area after the workspace was entered, the scheduler should replan accordingly.

The risk of sensor and communication failure should be taken into consideration to avoid costly failures. If a communication connection is lost, for example due to a broken cable, operators should be notified and the parts of the system that rely on that communication channel should stop while the rest of the system stays active, for example if the communication with a specific test station goes down that test station should be marked as unavailable until the communication is back up. The scheduler should replan accordingly. When checking for test units at the different positions within the system, either two redundant sensors or a sensor with integrated redundancy should be used. If these sensors give conflicting information, the operators should be notified and the parts of the system relying on that position should be inactivated, for example a test station or a position in the buffer area, until reactivated by an operator. The scheduler should replan accordingly. During testing, the system should be able to handle not only failed and passed test but also errors in the test station and errors in the pallet that are noticed during a test. If it is noticed that a test station is experiencing an error, operators and the scheduler should be notified and the test unit should be moved to a different test station of the same type when one is available. If an error in a pallet is noticed, the unit's test cycle should be aborted and the operators should be notified. It should be possible for the operators to distinguish between test units that have failed a test and test units that have experienced an error in its pallet.

2.5 Deadlocks

The system should prevent any avoidable deadlock situations. Deadlock situations that cannot be avoided are situations that occurs when a test cycle includes a type of test station that is not available or when the selected size of the buffer is to small to resolve the deadlock.

2.6 Data presentation

For each optics module the test time should be stored and available both locally and remote, since it is used by the billing system. It should be possible to locally visualize the current queue and the status of the modules within the system, regarding performed and upcoming tests as well as the estimated time until the module exits the system.

2.7 Statistics

To enable the possibility to perform a statistical analysis, not only the test results for each type of test station in the test cycle for each optics module should be stored but also the identification of the specific test station where the test was performed, the specific pallet and fixing plate used during the test as well as the time it took to perform the test. Furthermore, it should be possible to link each specific component used in a test with their respective types. To monitor and analyse the production itself, the average waiting time for the test units and the utilization for each test station and the robot should be stored. The mentioned data should be provided to Axis's main database system for storage and for performing the statistical analysis.

Chapter 3

Concept generation

3.1 General concept overview

In this chapter, two concepts for the flow of information within the system are described. While both alternatives were developed by using the system specification, see chapter 2, the first alternative, section 3.2, is to a larger extent purely based on the knowledge and experience of the student while in the second alternative, section 3.3, the ideas and visions of the project group at Axis had a bigger impact. Both concepts were developed iteratively with feedback from the project group. The two concepts are named the database concept and the ID-tag concept. However, both concepts utilize both databases and ID-tags. The name describes which technique is more heavily used in the one concept compared to the other concept. Section 3.2 describes the database concept completely while 3.3 describes how the ID-tag concept differs from the database concept.

Both concepts have the same physical flow of test units, see figure 3.1. In the assembly area, each optics module is connected to a fixing plate and a pallet forming a test unit, as described in section 2.1. The test unit is then sent to the pretest area where manual tests, such as connection tests and a visual inspection, are performed. If the test unit fails a test it will either be fixed in the pretest area or, in the case of more time consuming cases, it will be sent to the area for units that have not passed their test cycles. If the test unit passes the pretests it will be sent to the arrivals area from where it can be picked up by the robot. This area have two separate positions, both reachable by the robot. The robot will move test units from the arrivals area to the different test stations according to the specified test cycle of each test unit. In between tests it is possible for the robot to place and store test units in a buffer area to free up test stations for other test units. If a test unit either fails a test, if an error occurs within the test unit or if the test unit finishes an unofficial test cycle, i.e. a test cycle that does not fulfill the full requirements, the robot will move the test unit to the area for test units that have not passed the tests. If, however, a test unit passes all tests within a test cycle that fulfills the requirements the robot will move it to the area for passed test units. In the area for test units that have not passed the tests the test units will be repaired or, in more severe cases, discarded. Thereafter, the optics module will be disconnected from the pallet and the fixing plate before the three units are sent back to the assembly area, or the test unit will be sent back to the pretest area while still having the three components connected to each other. If an emergency stop is activated, if the entrance to the workspace of the robot has been opened or if the two openings in the test station are open at the same time, a safety PLC, Programmable Logic Controller, stops the robot, the conveyor and the test stations.

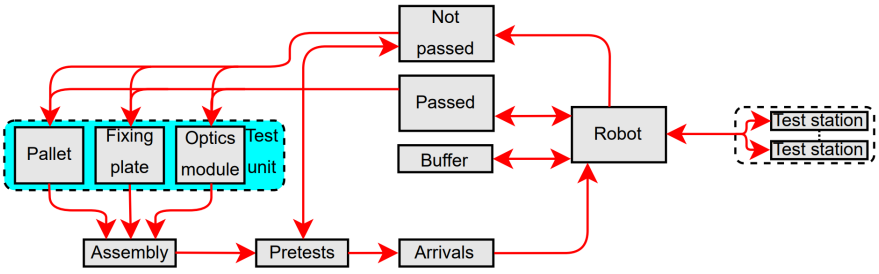

Figure 3.1: The physical flow of test units within the system.

3.2 The database concept

The database concept is visualised in figure 3.2. The messages within the system are described in tables 3.1, 3.2 and 3.3. It is recommended to follow the figure while reading this section. Each pallet stores an identification number and the current battery level on an ID-tag. The pallet updates the stored battery level at a certain frequency. Each fixing plate and optics module stores an identification number and a shared type number on ID-tags.

3.2.1 Preparation of the test units

In the assembly area, the battery level of the pallet is read to ensure the pallet is fit for use. The identification numbers of the pallet, fixing plate, and optics module are read, paired together, and saved to a secondary database, a database used only for this system which is quicker to access than the main database. The type number is read and sent with a request to the main database which in turn provides the assembly area with the test definition for the selected type number. The test definition describes all possible tests that the specified type can undertake, it states whether the tests need to be performed in a certain order or not, and it provides a standard test cycle to undertake. The operator at the assembly area decides on a test cycle and the chosen test cycle is sent to the secondary database along with the identification number of the fixing plate. In case the chosen test cycle consists of a test that is provided by a test station which is not currently active within the system, the operator will be presented with a warning. The list of available types of test stations is received from the scheduler. The secondary database also stores a list of all the types of optics modules and their priority number in the job scheduling. The operator can then choose to set a different priority for a specific test unit by having the new priority saved together with the identification number of the fixing plate in the secondary database.

The assembly area also receives information regarding self tests of the test stations from the scheduler. The operator first receives a warning and later, if no action has been taken, a final notice that the test station is unavailable until a self test has been carried out. The message consists of the type of test station and its position in the system, making it possible for the operator to make an informed decision of when to carry out the self test, depending on the current production flow. To carry out the self tests the operator will load a special test unit into the system. This test unit is only used for the self testing of the test stations. Finally, the assembly area receives a connection check from the main database, the secondary database and the scheduler. If a connection is lost, the operator at the assembly area is informed.

The test unit is then sent to the pretest area. Here the battery level of the pallet can be read since the information can be used while troubleshooting in case a pretest fails.

3.2.2 Arrival and buffer

In the arrivals area and the buffer area, the identification number and type number of the fixing plate are read. This information is sent to the scheduler together with the position of the test unit, determined by proximity sensors. If a proximity sensor in one of the areas is experiencing an error, that information will be shared with the scheduler. Finally, the scheduler will also receive a connection check from both areas.

When receiving an identification number and type number of a fixing plate from the arrival area, the scheduler sends these in a request to the secondary database that returns the earlier stored test cycle, the priority number for that type number and, if applicable, a specific priority number for that identification number as well as the estimated times in each test station within the test cycle.

3.2.3 Testing

For every test that is performed the scheduler will send the time duration to the secondary database together with the type number of the fixing plate and the type number of the test station where the test was carried out. If the estimated time and the actual times differ above a certain threshold the scheduler will request this list of test times for a given type of fixing plate and test station, recalculate the estimated time, and then send the new estimation to the secondary database for storage. To enable the possibility to share the time estimations between sites, the main database can send a request for the time estimate for a given type of test unit and test station. The main database can also send an updated time estimation for a given type of test unit and test station to the secondary database. Additionally, the main database can delete types of test units and test stations from the secondary database when the models have become deprecated. The secondary database has a connection check from the main database. If this connection is lost, the secondary database informs the scheduler. Furthermore, the scheduler has a connection check from the secondary database.

The testing stations have different configuration angles for different types of test units. To be able to take this into account in the planning, the scheduler requests the configuration angle for each type of test station and test unit as well as the angular velocity when changing configuration angle in each type of test station.

When a test station is being connected to the system it sends its type number, position and current configuration angle to the scheduler. If there already is a test unit in the test station at this point it will also send the identification number of the fixing plate and the type number of the test unit. The test stations will also send a request, containing their type number, to the main database to receive the correct configuration angles for each type of test unit. During the planning phase, the scheduler sends notices to the test stations telling the type of the next test unit that is planned for the test station, allowing the test station to configure accordingly. The notice also consists of a variable that tells whether the test station is the last in the test cycle or not and the identification number of the fixing plate. The identification number is used here to ensure that the received information concerns the same test unit as the one later received at the test station. If the test station is the last one in the test cycle it turns off the pallet after the test has been finished to save battery power.

When a test station receives a test unit it reads the identification numbers of the pallet, fixing plate and optics module as well as the battery level of the pallet and the type number of the test unit. The test station sends the identification number of the optics module to the main database as a request to receive any potential parameters derived in earlier tests. After a test has been carried out the test station informs the scheduler by sending the type number of the test station, the identification number of the fixing plate, the basic result, i.e. failed, passed, or error, the battery level of the pallet, and the time duration of the test. The test station also sends the three identification numbers and the type of the test unit, the battery level of the pallet, the identification number and position of the test station, the time duration of the test, any potential parameters needed for tests further down the test cycle, and the detailed result of the test to the main database for storage. Furthermore, the test stations inform the scheduler if they are experiencing an error, if the connection between the main database and a test station is lost, and if the front opening, where the robot enters, or the back opening, used for maintenance work, has been opened. The test stations also inform the scheduler when they are in need of a self test. These messages are not only forwarded to the assembly area, as described earlier in this section, but also to the control panel.

3.2.4 Visualizations and statistics

For visualisation purposes, the control panel receives the current queue within the system, the progress of the individual test units within the system, the latest reading of the battery level of the pallet for each test unit within the system and the estimated remaining time for each test unit within the system from the scheduler. The test units are identified using the identification number of the fixing plate. Additionally, the scheduler provides the control panel with the average utilization factor of the robots and the test stations as well as the average waiting time for the different types of test units. To change the priority number of different types of test units, the operator can request a list of the current priorities from the secondary database through the control panel. Thereafter the operator can send a new priority for a specific type of test unit back to the secondary database. To notify the operator regarding errors through the control panel, the scheduler sends one message to signal a connection error between two nodes and one message to signal an error within a node, for example a lost connection between a test station and the scheduler or an internal error in a test station. The control panel also has a connection check from the scheduler. To disconnect a test station from the system the operator sends a request from the control panel to the scheduler. The request consists of the position of the test station the operator want to access. The scheduler will replan accordingly and send a deactivation message to the test station in question. Thereafter, the scheduler will send a confirmation back to the control panel, saying that the access has been granted. When the operator wants to activate a test station or any node that has previously been excluded from the planning due to an error, for example a certain position in the buffer, the operator sends an activation message through the control panel to the scheduler. The scheduler will then replan accordingly and, in case the node to be activated is a test station, send an activation message to that test station.

The scheduler is not only providing the control panel with the average utilization factor of the robots and the test stations as well as the average waiting time for the different types of test units, but the information is also sent to the main database for storage. The scheduler also provides the main database with error logging, logging of emergency stops of the system, and logging of replanning situations due to test units within the system being moved by operators. As other connections, the scheduler has a connection check from the main database.

3.2.5 Safety

When an emergency stop has been activated, the entrance to the workspace of the robot has been opened, or both the front and back opening of a test station has been opened, the Safety PLC shuts down the conveyor, the robot and the test stations as well as informs the scheduler of what triggered the event. This information is forwarded to the control panel. Once the emergency stop, entrance or opening has been restored, the operator can start the system from the control panel again. The exact procedure of how to restore these events, which is needed to comply with safety regulations, are not described within this thesis.

3.2.6 The robot

To move test units from one position in the system to another, the robot receives commands from the scheduler. The scheduler will tell the robot to either pick a test unit at a certain position, to place a test unit at a certain position, or to go to a certain position without picking or placing a test unit. The scheduler can also command the robot to stop in the middle of one of these actions and the robot will inform the scheduler if the action is successful. To know the initial state of the robot, the robot informs the scheduler if it has a test unit in its gripper or not. Furthermore, the scheduler has a connection check from the robot. To avoid collisions when placing a test unit in the area for passed units or the area for units that have not passed the tests, these areas send a message to the robot telling whether the placing area is occupied or not.

3.2.7 Exiting the system

When a test unit arrives to the area for passed units or the area for units that have not passed their tests, the three identification numbers of the test unit are read. In the area of units that have not passed their tests the identification number of the fixing plate is compared to the corresponding identification number sent from the scheduler together with the basic result of that test unit. The basic result is used to distinguish between units that have failed their tests, units that have experienced an error and units that have completed an unofficial test cycle. Both areas use the identification number of the optics module in a request to the main database to receive the detailed result. The detailed result is used for troubleshooting failed test units and for manual verification of the result, a feature often appreciated by operators. As a final verification, the identification number of the fixing plate is sent to the secondary database, which returns the three identification number that were paired with it in the assembly area. These numbers should still be the same. Units that have not passed their tests are either sent back to the pretest area or the assembly area. Pallets and fixing plates from units that have passed their tests are sent back to the assembly area. When units are sent back to the assembly area, from either of the two areas, the pairing of the three identification numbers are deleted from the secondary database. It is not mandatory to detach the pallet from the fixing plate when sending the units back to the assembly area.

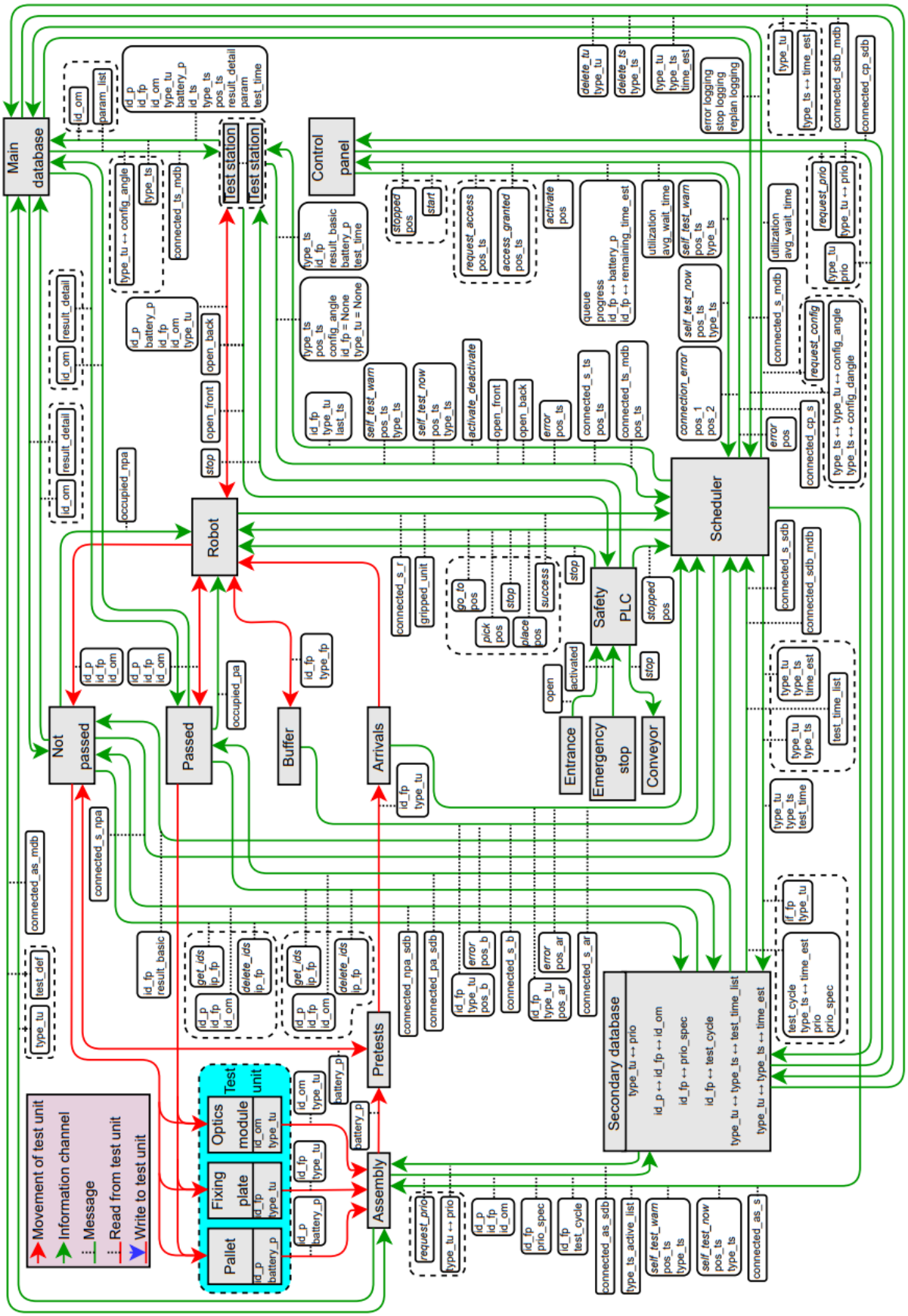

Figure 3.2: Flow of information within the database concept. Main power switch not included.

Message	Description
access_granted	A test station can now be manipulated manually.
activate	Activate a test station or position in the buffer or arrivals area.
activate_deactivate	Activate or deactivate a test station.
activated	Signals an activated emergency stop.
avg_wait_time	The average waiting time for each type of test unit.
battery_p	The battery level of the pallet.
config_angle	The configuration angle of a test station.
config_dangle	The angular velocity when changing configuration angle.
connected_as_mdb	Connection check from the main database to the assembly area.
connected_as_s	Connection check from the scheduler to the assembly area.
connected_as_sdb	Connection check from the secondary database to the assembly area.
connected_cp_s	Connection check from the scheduler to the control panel.
connected_s_ar	Connection check from the arrivals area to the scheduler.
connected_s_b	Connection check from the buffer to the scheduler.
connected_s_mdb	Connection check from the main database to the scheduler.
connected_s_npa	Connection check from the not passed area to the scheduler.
connected_s_r	Connection check from the robot area to the scheduler.
connected_s_sdb	Connection check from the secondary database to the scheduler.
connected_s_ts	Connection check from a test station to the scheduler.
connected_sdb_mdb	Connection check from the main database to the secondary database.
connected_ts_mdb	Connection check from the main database to a test station.
connection_error	Signals a connection error between two subsystems.
delete_ids	Delete the paired identification numbers.
delete_ts	Delete a type of test station from memory.
delete_tu	Delete a type of test unit from memory.
error	Signals an error at a test station, in the arrivals area, or in the buffer.
error_logging	Logging the errors that have occurred.
get_ids	Get the stored paired identification numbers.
got_to	Command the robot to go to a specific position.
gripped_unit	States whether the robot is holding a test unit or not.
id_fp	The identification number of the fixing plate.
id_om	The identification number of the optics module.
id_om_paired	The identification number of connected optics module.
id_p	The identification number of the pallet.
id_p_paired	The identification number of connected pallet.
last_ts	States whether a test station is the last one in a test cycle or not.
occupied_npa	States whether the not passed area is occupied or not.
occupied_pa	States whether the passed area is occupied or not.
open	States whether the entrance to the robot's workspace is open or not.
open_back	States whether the back of a test station is open or not.
open_front	States whether the front of a test station is open or not.
param	Parameters derived during a test.
param_list	A list of all parameters derived for a specific optics module.
pick	Command the robot to pick a test unit at a specific position.
place	Command the robot to place a test unit at a specific position.
pos	A position in the system.
pos_ar	A position in the arrivals area.
pos_b	A position in the buffer.
pos_ts	A position of a test station.
prio	The prioritization number for a type of test unit.
prio	The prioritization number for a specific test unit.
prio_spec	The prioritization number for a specific test unit.
progress	The performed and remaining test for each test unit in the system.
queue	The currently planned schedule.

Only in the database concept
Only in the ID-tag concept

Table 3.1: Description of the messages within the two concepts, part 1.

Message	Description
remaining_time_est	The estimation of the remaining time in the system for a test unit.
replan_logging	Logging the replanning that have occurred due to unexpected movements of test units.
request_access	Request access to manipulate a test station.
request_config	Request the configuration angle and angular velocity, for each type of test station and test unit.
request_prio	Request the prioritization number for each type of test unit.
result_basic	The basic result of a test, i.e. passed, failed, or error.
result_basic_list	A list of the basic results from all tests for a test unit.
result_detail	All details concerning a test result for a test unit.
self_test_now	Indicates that a self test of a test station should be performed now.
self_test_warm	Indicates that a self test of a test station should be performed soon.
start	Restart the system after an emergency stop.
stop	Commands the robot, the test stations and the conveyor to stop.
stop_logging	Logging the emergency stops that have occurred.
stopped	States that the system has been stopped due to a safety violation at a certain position.
success	States whether the action of the robot was successful or not.
test_cycle	The chosen test cycle for a specific test unit.
test_def	A definition of the possible test cycles for a specific type of test unit.
test_time	The time duration of a test.
test_time_list	A list of all test times for a type of test unit and test station.
time_est	The estimation of the test time for a type of test unit and test station.
type_ts	The type number of a test station.
type_ts_active_list	A list of all types of test stations that are currently active.
type_tu	The type number of a test unit.
utilization	The utilization factors of the robot and the test stations.

Only in the database concept
Only in the ID-tag concept

Table 3.2: Description of the messages within the two concepts, part 2.

Message extension	Description
↔	A list of paired up variables.
= None	Only included if currently available.

Table 3.3: Description of the message extensions within the two concepts.

3.3 The ID-tag concept

It is recommended to follow figure 3.3 while reading this section. As for the database concept, the messages within the system are described in tables 3.1, 3.2 and 3.3. The fundamental design of the ID-tag concept is to a large extent equal to the design of the database concept, see section 3.2. However, the features that makes the ID-tag concept differ from the database concept are described in this section. In this concept there is no secondary database. Instead, additional information is stored on the fixing plate. This additional information consists of the identification numbers of the paired pallet and optics module, the chosen test cycle, the chosen priority number as well as a list of the basic results.

In the assembly area, the test cycle, priority number and identification number of the paired pallet and optics module are written to the fixing plate instead of stored in a secondary database. The scheduler receives the test cycle and priority number directly from the arrivals area, buffer area, or test stations, the later two in case there is already a test unit in the buffer area or in a test station when the system is activated. The basic result from each test station is both sent to the scheduler and written to the fixing plate. Like this, the area for units that have not passed their tests can directly read the basic results from the test unit instead of receiving it from the scheduler. Similarly, both the area for units that have passed their tests and the area for the units that have not passed their tests can read the identification number of the paired pallet and the paired optics module and thereafter delete them instead of contacting a secondary database. Here, the basic result also has to be deleted from the test unit before the fixing plate enters the system again.

Since this concept does not include a secondary database the management of time estimations for the tests in the different types of test stations need to be solved differently in this concept compared to the database concept. It was chosen to use the main database for this. The scheduler sends a request with the type of test unit to the main database. The response from the database consists of the estimated times for each type test station for that type of test unit. In case the actual times differ above a certain threshold the scheduler will request a list of test times for a given type of test unit and test station, recalculate the estimated time, and then send the new estimation to the database for storage. Since the test times are included in the result message from the test stations to the main database, the database already has access to those.

Additionally, this concept does not include standard priorities for the different types of optics modules since this feature too was enabled using the secondary database in the database concept. Instead, the priority that is chosen for the first optics module of each type since startup at the assembly area is used as a standard value for that type until it is changed or until the system has been shut down.

The rest of the concept design is identical between the two concepts, i.e. the difference is whether to use a secondary database or to write more information to the test units themselves.

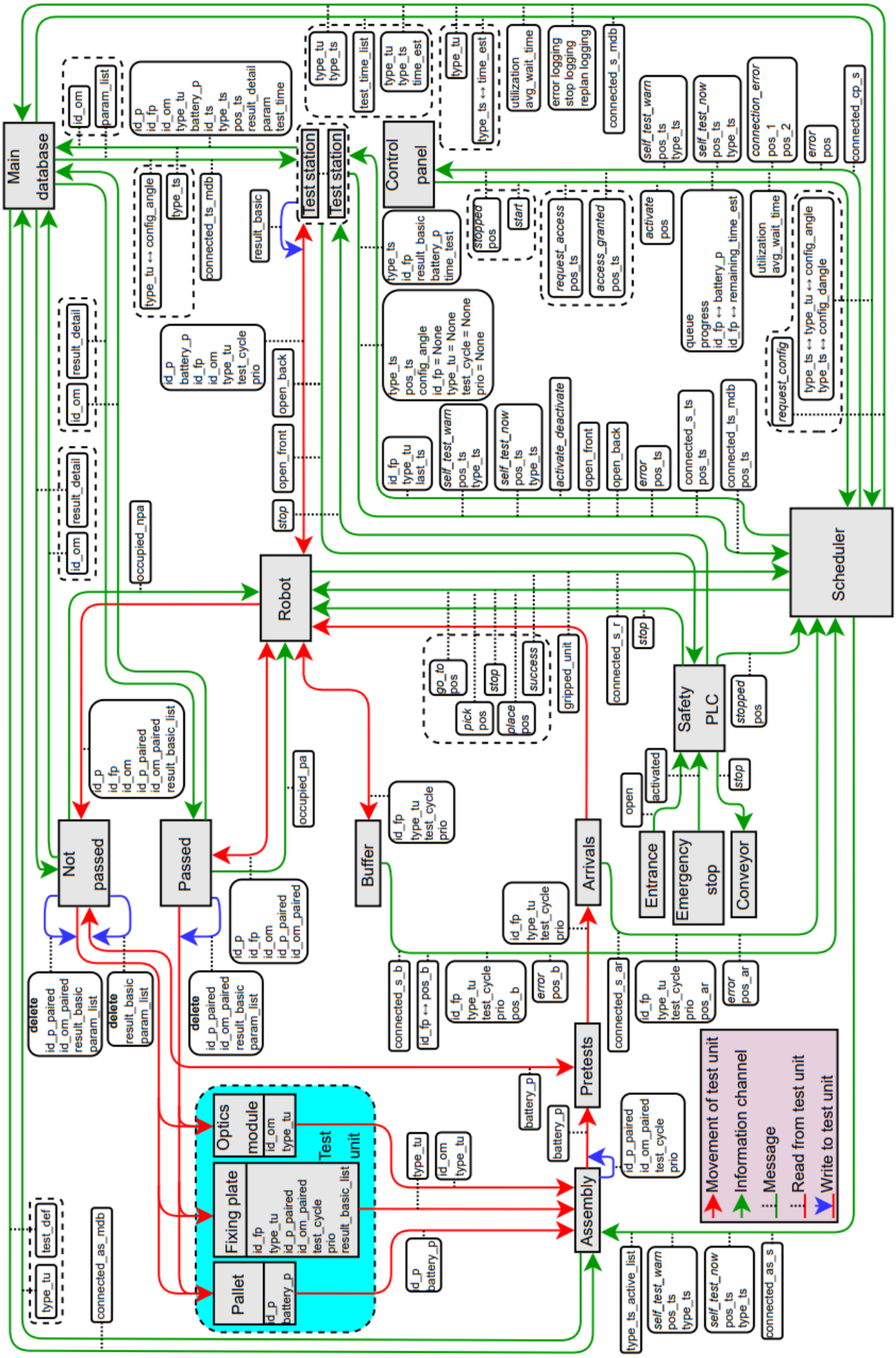

Figure 3.3: Flow of information within the ID-tag concept. Main power switch not included.

3.4 Summary of the communication channels and the essential hardware

This section summarizes a high level overview of the necessary communication channels and hardware for the flow of information within the two concepts. Details such as recommending specific products or brands are not covered. Instead the focus is on general functionality which leaves the future implementation open for adjustments to fit in with Axis's other systems, potential contracts with specific resellers, the desires of the company to be hired for the system integration and any future changes of the system. However, the components that are eventually chosen need to comply with the description in this section. The components that are necessary for the conveyor, for performing the different tests in the test stations and in the pretest area as well as the tools required for the physical assembly and dismantling is not included. For a summary in the form of a table, see table 3.4 for the essential hardware and table 3.5 for the communication channels.

It should be possible to read stored information from the pallets, fixing plates and optics modules. In the ID-tag concept, it should also be possible to overwrite the information on the fixing plate. Contactless data transfer would simplify this automatic procedure. For the pallet and fixing plate, RFID-tags can be used. To lower the cost of the product, the optics modules can use QR-codes instead. Additionally, it should be possible to measure the remaining battery power of the pallet. This can be done by having the pallet itself update the stored battery power on its RFID-tag at a certain rate.

In both concepts, the assembly area needs to read for the information stored on the test units. In the ID-tag concept, writing functionality is also needed. When it comes to communication, the assembly area needs to have a communication channel to and from the main database and from the scheduler. In the database concept communication to and from the secondary database is also needed. To visualize the received data and then manipulate it before it is transmitted, an HMI, Human Machine Interface, is needed in this area. Next up is the pretest area. In both concepts, this area only needs to read the remaining battery power of the pallet and present it on an HMI. If the assembly area and pretest area is constructed together as one area, they could share the same hardware.

The arrivals area needs a reader for the information stored on the fixing plate. It also needs one proximity sensor for each position within the area. The data from the proximity sensors needs to be processed and sent together with information from the fixing plate to the scheduler.

The robot not only needs the hardware and software necessary to run, but also a proximity sensor in its gripper and a communication channel to and from the scheduler. Furthermore, the robot needs to receive data from the safety PLC, the area for passed units and the area for units that did not pass. These three communication channels can be more simplistic compared to the communication channel between, for example, the robot and the scheduler, since this information consist of booleans, i.e. each of these messages can be represented by a signal that is either high or low, rather than more complex data structures. The area for units that passed and the area for units that did not pass both need to read the information on the pallet, fixing plate and optics module. In the ID-tag concept, the ability to write new information to the fixing plate is also needed. Furthermore, the areas need communication to and from the main database. In the database concept, the area for units that did not pass needs communication to and from the scheduler and both the areas need communication to and from the secondary database. Additionally, both areas need a proximity sensor and an HMI, the latter for visualizing the results of the test units.

Continuing with the buffer, this area needs a reader for the information on the fixing plate and a proximity sensor for each position within the area. The data from the proximity sensors needs to be processed and sent together with information from the fixing plate to the scheduler.

The control panel needs a communication channel to and from the scheduler. In the database concept, it also needs a communication channel to and from the secondary database. Additionally, as the name suggests, this area needs an HMI to visualize and manipulate the information.

Moving on to the test stations. The test stations need to read the information from not only the fixing plate but also the pallet and optics module. In the ID-tag concept, the ability to write information to the fixing plate is also needed. When it comes to communication channels, the test stations need channels to and from the main database and the scheduler. Additionally, the test stations communicates to and from the safety PLC, however, this communication can be more simplistic since the information consist of booleans, a signal that is either high or low, rather than more complex data structures.

Additionally to the robot and the test stations, the safety PLC also needs communication channels to the scheduler and the conveyor as well as communication channels from the entrance to the robot's workspace and from the emergency stops in the system. Here, all communication channels but the communication channel to the scheduler can be of a more simplistic nature since, again, the information consists of booleans, a signal that is either high or low, rather than more complex data structures. The communication channels of the scheduler and the main and secondary database have already been described. The secondary database is only included in the database concept. This database can either be completely separated from Axis's main database management system or integrated within that system.

The communication channels that have been described as more simplistic in this section can be implemented using I/O-communication. In case a signal is lost, a safe state should be assumed. For proximity sensors, the safe state is to assume that the position is occupied. For safety messages used for shutting down the system, the safe state is to assume that the system should be shut down. The rest of the communication channels uses more complex data structures which require a higher bandwidth. These channels should include a verification of the status of the channel, i.e. if the channel is working as intended or not. These communication channels could be implemented using for example Ethernet.

As stated in chapter 2, all proximity sensors should include redundancy for increased reliability. Shawn Frayne [3] describes four common types of proximity sensors: IR-sensors, ultrasonic sensors, capacitive sensors and inductive sensors. The inductive sensor can only sense metal, it has a short sensing range, it is suitable for harsh environments and it is in general the cheapest sensor out of these four. The three other sensors that are mentioned have the advantage that they can also sense other types of materials. Additionally, ultrasonic sensors and IR-sensors have a longer sensing range. However, ultrasonic sensors work best when detecting flat surfaces and IR-sensors are sensitive to, for example, dust blocking the beam of light [3]. In this system, all proximity sensors senses whether a test unit is present or not at a specific location. The shape and material of the test units can be adapted to fit with the selected type of sensor, for example the test unit can either be made out of metal or have a metal plate attached to it if this is necessary for the selected type of proximity sensor. The system can also be designed so that the test units are being placed right by the sensor, meaning that a long sensing range is not required. The system will be working in an industrial environment but since dust can cause issues for the optics modules, the cleanliness of the working environment have certain requirements. Taking all this into account, inductive sensors are recommended. They comply with the requirements of the system while at the same time, in general, being cheaper than the three other types of sensors that are discussed.

Subsystem	Hardware
Pallets	RFID-tag, RW RFID-reader MCU
Fixing plates	RFID-tag, WORM RFID-tag, RW
Optics modules	QR-code
Assembly	RFID-reader QR-reader MCU HMI
Pretests	RFID-reader MCU HMI
Arrivals	2 RFID-readers 2 inductive sensors MCU
Buffer	1 RFID-reader per pos. 1 inductive sensor per pos. MCU
Passed	RFID-reader QR-reader Inductive sensor MCU HMI
Not passed	RFID-reader QR-reader Inductive sensor MCU HMI
Robot	Industrial robot Robot controller Gripper Inductive sensor
Test stations	2 RFID-readers QR-reader MCU
Control panel	MCU HMI
Safety PLC	Safety PLC
Scheduler	PLC
Main database	Database DBMS
Secondary database	Database DBMS

Only in the database concept
Only in the ID-tag concept

Table 3.4: A summary of the essential hardware for each subsystem.

Subsystem	Ethernet com. with	Input from	Output to	RFID/QR read from	RFID write to
Assembly	Scheduler Main database Secondary database			Pallet Fixing plate Optics module	Fixing plate
Pretests					
Arrivals	Scheduler			Fixing plate	
Buffer	Scheduler			Fixing plate	
Passed	Main database Secondary database		Robot	Pallet Fixing plate Optics module	Fixing plate
Not passed	Main database Secondary database Scheduler		Robot	Pallet Fixing plate Optics module	Fixing plate
Robot	Scheduler	Passed Not passed Safety PLC			
Test stations	Scheduler Main database	Safety PLC	Safety PLC	Pallet Fixing plate Optics module	Fixing plate
Control panel	Scheduler Secondary database				
Safety PLC	Scheduler	Test stations Emergency stops Entrance	Test stations Robot Conveyor		
Scheduler	Assembly Arrivals Buffer Not passed Robot Test stations Control panel Safety PLC Main database Secondary database				
Main database	Assembly Passed Not passed Test stations Scheduler Secondary database				
Secondary database	Assembly Passed Not passed Control panel Scheduler Main database				

Only in the database concept
Only in the ID-tag concept

Table 3.5: A summary of the communication between the different subsystems.

3.5 Comparison of the concepts

Both concepts to a large extent offer the same functionality in the form of capabilities, time efficiency and the available information at each area. The only exception to this is the handling of estimated test times and standardized priorities.

In the database concept, estimated test times are accessed and updated through the secondary database while in the ID-tag concept they are accessed and updated through the main database. Due to this, the process is slower in the ID-tag concept compared to in the database concept. However, the time estimations for a type of test unit only need to be accessed the first time that type of test unit is introduced to the system since startup and updates are only performed when the estimated times and actual times have a difference that is above a certain threshold.

In the database concept, a standard priority value for each type of optics module is stored in the secondary database and can be changed using the control panel. In the ID-tag concept, the standard priorities are not stored after the system has been shut down. Instead, the priority that is chosen for the first optics module of each type since startup at the assembly area is used as a standard value for that type until it is changed or until the system has been shut down.

What differs more is the implementation. The ID-tag concept uses fewer communication channels since more information is transferred with the test units. Additionally, the database concept uses a secondary database that results in a more complex database management system compared to the ID-tag concept. On the other hand, the ID-tag concept puts a higher demand on the hardware that is used to store information on the fixing plates since, in this concept, this information is being overwritten several times during each cycle in the system and, compared to the database concept, more information is stored.

Another aspect is the potential for future updates and expansions of the system. Here the complexity of the database concept provides an advantage over the ID-tag concept. Since more areas are connected in the database concept, more complex features can be added without updating the hardware. However, it is also an option to include these connections in the ID-tag concept, without utilizing them in the first version of the system, to enable future usage through software updates.

Taking just this into account, the ID-tag concept would be recommended for a future implementation since it offers similar capabilities with a system design that is less complex. However, when discussing the two concepts with the team at Axis, the team conveyed that the structure of the database concept is more in line with their current expertise than the ID-tag concept. It was estimated that the advantages of the ID-tag concept are not significant enough to overlook this aspect and therefore the database concept is recommended for a future implementation.

Chapter 4

Job shop scheduling

In this chapter, a job shop scheduling method for the system is derived and simulated. First, in section 4.1, a scheduling model for the system is defined, in section 4.2, theories and methods regarding job shop scheduling are described and in section 4.3, the theories are applied on the system concerned in this thesis.

4.1 Scheduling model

In the system for this project, test units, in regards of scheduling referred to as jobs, j_i, arrives to the arrivals area, an input buffer, B_0, with the size of two. The jobs will then go through a defined set of test stations with the help of a robot. In which order a certain job should go should go through the specified test stations can be either specified, chosen freely or partly specified. Both the robot and the test stations are here referred to as machines, M_0 being the robot and M_1 to M_n being the test stations. The time duration for a job, j_i, in a test station, M_k, i.e. the processing time, can be referred to as $p_{M_k j_i}$. The system also includes a buffer, B_1, where jobs can be placed by the robot to free up test stations or the input buffer. In this project, the buffer will have a limited size which can be varied in between simulations. Once jobs are finished or cancelled, due to errors or failures, they leave the system through one of the output areas, with the help of the robot. Each of the test stations, M_1 to M_n, has a certain setup time when changing from one type of job to another. This is due to different jobs requiring different configuration angles in the test stations. The setup can be performed as soon as the previous job has been processed, i.e. the previous job does not have to leave the machine before the setup can begin. The setup time between jobs, j_i and j_l, in a test station, M_k, can be referred to as $s_{M_k j_i j_l}$. There is also a setup time for the robot, the time to get from its current position to the machine or buffer where a job will be collected. This setup can also be performed beforehand, i.e. before the job is ready in a machine. The setup time for the robot when going from, for example B_i to M_k, can be referred to as $s_{M_0 B_i M_k}$. For the robot, an equivalent processing time can be defined as, in the cases where the pick-up position is a machine, the time it takes to enter the machine, the time it takes to collect a job, deliver it to its next location and, in the cases where the drop-off position is a machine, have the robot exit the machine. If a job is for example collected as M_k and will be placed at B_i, the processing time can be referred to as $p_{M_0 M_k B_i}$. In addition to the machine and buffers, the robot have three more positions: the two output areas, referred to as O_i and its home position, referred to as H. In the cases where the robot is starting in another position than the ones described here, the robot will go to the home position and initialize the scheduling from there. The scheduler should be flexible when it comes to introduction of more test stations, deactivation of test stations, jobs in unpredicted initial starting positions and cancellations of jobs. All specified time durations will be approximated, i.e. the true times may deviate from the approximations. The described model is visualised in figure 4.1.

In this system, jobs do not have a specified due date, it should however be possible to give different jobs different priorities. The main objective is to maximize the throughput of the system while no jobs get stuck in the system waiting.

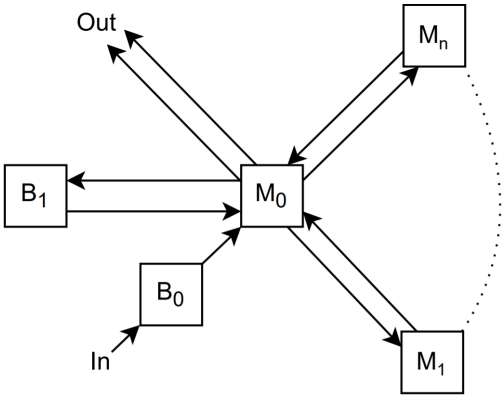

Figure 4.1: The scheduling model.

4.2 Job shop scheduling theory

For static scheduling problems consisting of one to two machines and n jobs, for cases with n machines and two jobs and, under certain conditions, for problems consisting of three machines and n jobs as well as for a limited number of special cases of more complex systems, general methods have been derived. However, optimizing the general case of more complex systems are NP-hard [10]. A problem is NP if it is solvable in nondeterministic polynomial time and if a problem is NP-hard it means that an algorithm for solving it can be translated into one that solves any other NP-problem [13]. None of the cases that have a general method derived matches the dynamic model described in section 4.1.

For dynamic cases like this, with new job arrivals, machine breakdowns and job cancellations, Wang et al. [12] present a framework for how to handle dynamic job shop scheduling. They divide dynamic scheduling into four areas: methods, strategies, policies and approaches, see figure 4.2.

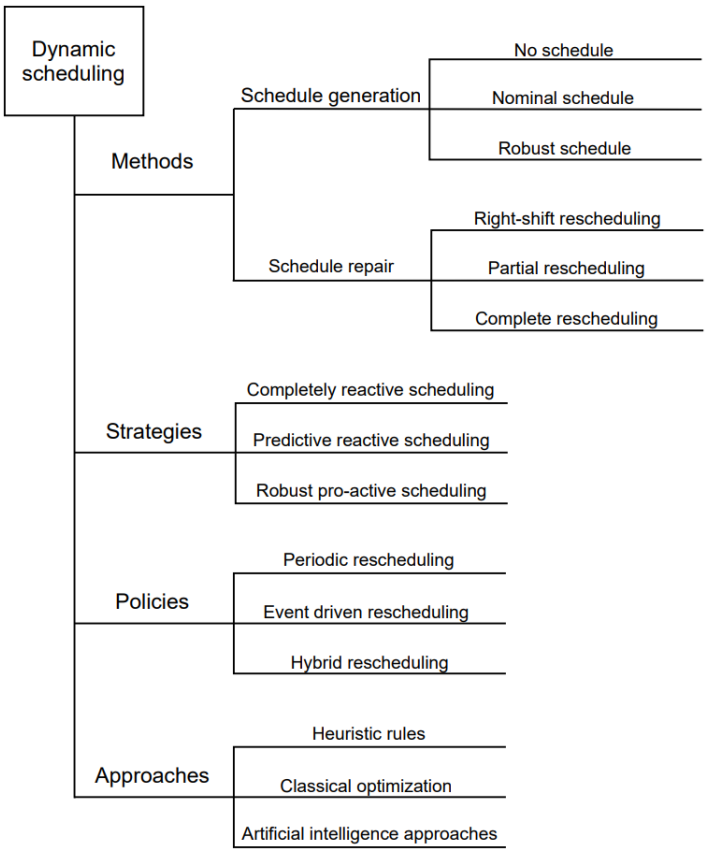

Figure 4.2: The scheduling framework presented by Wang et al. [12].

The methods are split up into two categories, schedule generation and schedule repair. For schedule generation three methods are mentioned, no schedule, nominal schedule and robust schedule. With a nominal schedule, the initial schedule is created without real time events, such as job cancellation, taken into account. With a robust schedule, the initial schedule builds upon predictions of upcoming real time events. In the final case, no schedule, no initial schedule is generated. Instead commands are chosen in real time. Wang et al. also name three alternatives when it comes to schedule repair, right-shift rescheduling, complete rescheduling and partial rescheduling. In right-shift rescheduling, the content of the schedule is pushed forward in time to allow for unexpected time delays. In complete rescheduling, the whole schedule is reevaluated once a real time events affect the schedule. Finally, in partial rescheduling, only the operations that are affected by a real time event are reevaluated [12].

Wang et al. describe three different strategies. Completely reactive scheduling, predictive reactive scheduling and robust pro-active scheduling. In completely reactive scheduling the scheduling is performed in real time without predictions. In predictive reactive scheduling, a schedule is generated by predicting the outcome of a system. The schedule is then adapted according to real time events. In robust pro-active scheduling, the effect of system disturbances and performances are predicted beforehand to generate a schedule in advance [12].

Furthermore, Wang et al. present three policies for implementing the strategies: periodic rescheduling, event-driven rescheduling and hybrid rescheduling. All three policies take real time events into account, however, periodic rescheduling takes events into account only at a specific rate, event-driven rescheduling handles the events as they occur and in hybrid rescheduling the rescheduling is done periodically but it is only carried out in case a real time event has occurred [12].

Finally, Wang et al. bring up three approaches, heuristic rules, classical optimization and artificial intelligence approaches [12]. These require a more substantial review and are therefore described in more detail in the upcoming subsections, namely subsection 4.2.1, 4.2.2 and 4.2.3.

4.2.1 Heuristic rules

Heuristic rules, also known as dispatching rules, are prioritization rules based on attributes of jobs, machines and time. In each step when choosing a job to process, the job with the highest resulting priority according to the rule is chosen. Depending on which factors the objective is related to, factors such as maximum lateness, variation in waiting time, workload balancing, throughput and so on, different rules tend to be more effective. However, no guarantees are given. [5]

Michael L. Pinedo [5] describes 12 standardized heuristic rules and for which objectives they tend to be the most effective.

- *The earliest release date first rule*, the job that arrived to the system first is chosen first, tends to minimize the variation in the waiting time.
- *The earliest due date rule*, the job with the earliest due date is chosen first, and *the minimum slack rule*, the job with the smallest difference between the time until the due date and the processing time is chosen first, tend to minimize the maximum lateness.
- *The longest processing time first rule* where, as the name suggests, the job with the longest processing time is chosen first, tend to balance the workload over the different machines to the maximum in cases were machines work in parallel.
- *The shortest processing time first rule* tends to minimize the sum of the completion times of the jobs.
- By modifying the shortest processing time first rule by introducing weights to the different jobs, *the weighted shorted processing time first rule* tends to minimize the weighted sum of the completion times.
- When the queue behind each available job is known, *the critical path rule*, where the job that is in first place of the queue with the highest sum of processing times is chosen first, and *the largest number of successors rule*, the job that has the highest number of jobs in queue behind it is chosen first, tend to minimize the makespan of the jobs, i.e. the time duration that each job spends within the system.

- *The shortest setup time first rule*, the job that requires the shortest setup time is chosen, and *the least flexible job first rule*, where the job that has the lowest number of alternative machines to chose from is chosen first, tend to minimize the makespan of the jobs and maximize throughput of the system.
- *The shortest queue at the next operation rule*, the job that has the shortest queue in between leaving the upcoming machine and entering the next one is chosen first, tends to minimize the machine idleness.
- Finally, *the service in random order rule*, where jobs are chosen at random, is the solution that tends to minimize the complexity of the implementation.

However, in practise the sought objective could be a time dependent combination of several factors. Pinedo [5] explains that several heuristic rules can be combined into more complex rules by applying scaling functions to a combination of the standardized heuristic rules. These rules have to be determined by the developer to fit the needs of the specific system. Experience is helpful when designing the rules but simulations are needed to fully validate them [5].

4.2.2 Classical optimization

When solving the scheduling problem by using classical optimization, dynamic programming and branch and bound methods are the most commonly used techniques [6].

Michael L. Pinedo [6] explains that the idea behind dynamic programming is to divide the problem into subproblems and calculate the optimal solution for each subproblem. When there are more than one way of dividing the problem, each possible way is explored. The solutions are then combined to form the main problem and an optimal solution for the main problem is calculated. Pinedo concretizes dynamic programming for the case of dynamic job shop scheduling. Each job is given a cost. The cost depends on the time when the job is finished and can include different scaling factors for different jobs. The sum of the costs for all jobs forms the full objective that should be minimized to achieve the optimal solution. The jobs that are available for processing are split into subgroups, each with a fixed number of jobs. When the total number of jobs is a prime number, one single group can consist of fewer jobs than the others. Dynamic programming has two different approaches, forward and backward. In forward dynamic programming, one subgroup is chosen as the first one to process. By iterating through all different combinations, the optimal order for the first subgroup, when it is processed first, is found. Then, another subgroup is chosen as the second subgroup to be processed and the optimal order for that subgroup, when processed as the second subgroup, is found in the same way as for the first subgroup. This goes on until an optimal order has been derived within each subgroup, given a set processing order of the subgroups. If one subgroup has fewer jobs than the other, this subgroup should be processed last. This process is repeated for every possible order of the subgroups. As a final step, this process is repeated for every possible formation of subgroups of the previously chosen fixed size. Once this has been done, the solution with the lowest total cost is the optimal processing order. Backward dynamic programming works similarly to forward dynamic programming. The only difference is that instead of first choosing the first subgroup to be processed and work from there, the last subgroup to be processed is chosen first and the method continues by working towards the first subgroup to be processed. If n jobs shall be processed in one machine, it can be shown that the total number of evaluations needed to solve the problem with this method are $O(2^n)$ [6]. A visualization of forward dynamic programming for job scheduling is shown in figure 4.3.

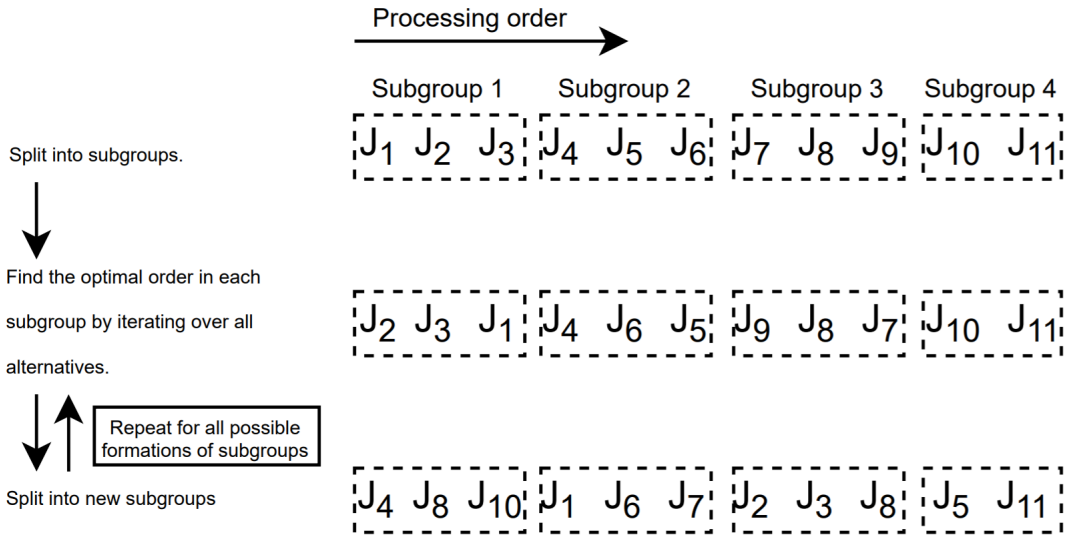

Figure 4.3: A visualization of forward dynamic programming for job scheduling.

Patrick Winston, instructor at MIT, explains the branch and bound method by applying it to an example where the shortest path between two nodes in a map is sought [14]. The paths between the nodes in the map each have a positive, nonzero, cost assigned to them. A search tree is built with the starting node as the root. Each possible path leading to new nodes from the root are explored, forming branches in the search tree. Once all paths that are leading out of the root have been explored, the algorithm steps down one layer in the tree and starts exploring all paths leading out of the second layer of nodes, starting with the node that has the lowest cumulative sum of costs in its path from the root. One node at a time, the algorithm continues to explore paths leading out of the nodes, always starting with the node with the least cumulative sum of costs, until the end node has been reached and no node with unexplored paths and a shorter cumulative sum of costs than the full way to the goal exists. By adding what is called an extended list to the branch and bound method, the algorithm keeps track of which nodes it has reached previously and what the lowest cumulative sum of costs from the root to each node is. If the algorithm comes back to a node it has already visited through another branch, only the branch with the lowest cumulative sum of costs will be expanded further [14]. The method is visualized with an example in figure 4.4.

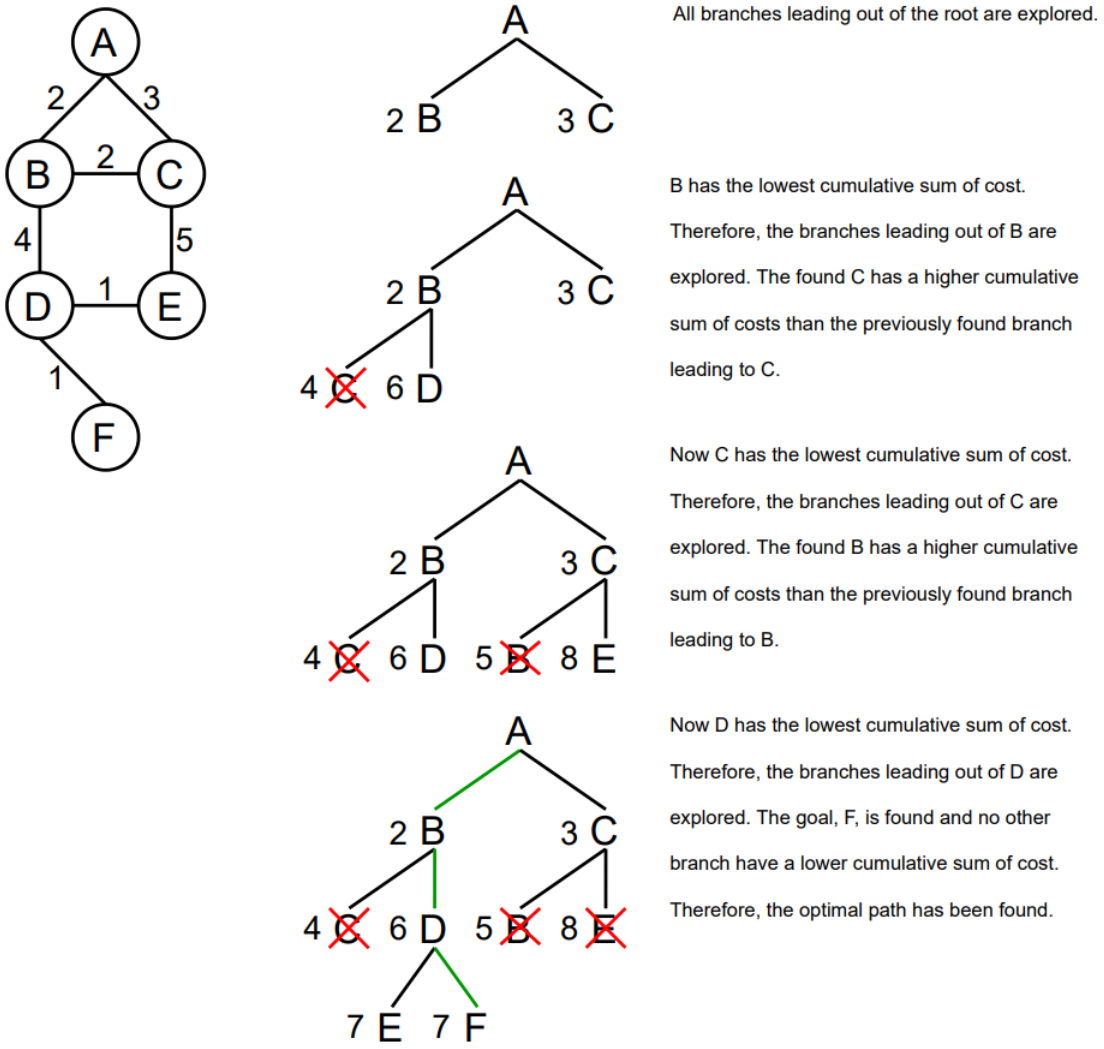

Figure 4.4: A visualization of the branch and bound method with an extended list. The optimal path from A to F in the map to the left in the figure is sought.

Furthermore, Winston [14] describes that in cases where the remaining distance from a node to the goal can be given a lower bound before the branches leading out from that node have been explored, the algorithm can be improved further. Instead of extending the node with the lowest cumulative sum of costs next, the node where the sum of the cumulative sum of costs and the lower bound of the remaining distance is the lowest, is extended first. When the goal node has been reached and, by taking the lower bound of the remaining distance into account, no other branch can reach the goal at a lower cost, the optimal path has been found [14].

The branch and bound method used together with an extended list and with utilization of lower bounds on the remaining distance from each node forms a method known as A*, pronounced A-star. Russell and Norvig [11] state that the time complexity of A* depends on the accuracy of the set lower bound, i.e. how close the lower bound is to the actual cost. However, for most heuristics in practical use, the time complexity is exponential in the solution depth of the search tree. The main drawback of A* is however not the time complexity, but rather the space complexity since all generated nodes are stored in memory [11].

The explanation of branch and bound and A* given by Winston is based on finding the optimal route between two nodes in a map [14]. For the purpose of this thesis, the problem needs to be reformulated to fit with job scheduling, a problem that has different characteristics depending on the application. For some special cases, the adaptation of the algorithm has received considerable attention in previous studies. One example of such special cases is the case with one machine and several jobs that each have a release date and a due date. The adaptations of the algorithm for this problem has been described by Pinedo [7].

4.2.3 Artificial intelligence approaches

The artificial intelligence techniques that are typical for solving the job scheduling problem are often meta heuristic approaches, such as the genetic algorithm, tabu search, beam search, multi-agent systems, neural networks, ant colony optimization, artificial bee colony algorithm and variable neighbourhood search [12]. To limit the scope of this analysis, only beam search, genetic algorithms and tabu search are covered here.

According to Pinedo, the beam search method combines the branch and bound method with heuristic rules, see subsections 4.2.1 and 4.2.2, to eliminate the amount of branches that are being extended [8]. At each level in the search tree, the nodes are evaluated and only the most promising nodes are extended further. As a compromise between spending time on extending more nodes and spending time on performing a more complex evaluation of the nodes, the evaluation process can be split up in two stages. First, a quicker evaluation is performed on all nodes at the current level. The nodes that perform the best in this evaluation move on to the next stage. The amount of nodes that moves on to the next stage is referred to as the filter width. In the second stage, a more time consuming evaluation method is performed on the remaining nodes and only the nodes with the best results are extended to the next level in the search tree. The amount of nodes that is further extended is refereed to as the beam width. The evaluation process consists of heuristic rules or a composition of several heuristic rules. This method decreases the computing time of the branch and bound method but there is no longer a guarantee that the found solution is optimal [8].

Genetic algorithms are also explained by Pinedo [8]. This algorithm uses some heuristic rules, see subsection 4.2.1, to create several schedules. The generated schedules are ranked according to a selected objective, for example the throughput of the system or the average waiting times of the jobs etc. The two schedules that performs the worst are disregarded and two new schedules are constructed by in a predefined way combining features from the two schedules that performed the best. The newly created schedules are considered neighbors to the schedules they originated from. In the example of parallel machines simultaneously processing a sequence of jobs each an example of a feature to combine is the sequences in different machines. The remaining schedules, that is all schedules but the two that performed the worst, are kept the same and the ranking process starts again. This process continues for a predefined number of iterations before the best performing schedule is chosen. Just as for the beam search method, this method does not guarantee an optimal result [8].

Tabu search is also described by Pinedo [8]. Tabu search can be seen as a special case of genetic algorithms. The algorithm is described in the following way. An initial schedule is generated using some heuristic rules. New schedules from the neighbourhood of the initial schedule are generated. If the system consists of several jobs that should be processed in a single machine, the neighbourhood can be defined as the schedules that can be generated by switching place on two adjacent jobs. Out of the newly generated schedules, the best performing schedule according to a selected objective function is kept and the change from the initial schedule to the chosen schedule, defined by for example the two jobs that switched place in the previously mentioned example, is kept in a so called tabu list. If the generated schedule performs better than the initial schedule, according to the same objective function, the new schedule is also stored as the best schedule so far. The algorithm performs another iteration by generating new schedules out of the neighbourhood of the previously generated schedule, regardless if the schedule was better than the last one or not. However, after the initial iteration the tabu list comes into place. At each iteration, changes that are already in the tabu list may not be performed again, meaning that schedules generated through such changes should not be considered when picking the best performing schedule within the neighbourhood. At each iteration the chosen change is added to the tabu list. However, the tabu list has a limited size so at each iteration the oldest element in the list will be deleted and it will once again be considered as a valid option. The tabu list is used so that the algorithm does not come back to the same local minimum over and over again. When the algorithm has gone through a set number of iterations or when a better performing schedule has not been found for a certain number of iterations, the algorithm is finished and the schedule that performed the best is applied. In more complex setups compared to the mentioned example, defining the neighbourhood of a schedule is a considerable challenge when applying the algorithm. As for the beam search method and genetic algorithms in general, this method does not guarantee an optimal result. There are also more complex versions of tabu search, for example one method that uses a so called tabu tree. Instead of just saving the tabu list and the best solution so far, nodes are generated in a tree structure, each level in the tree being the neighbourhood of the layer above. In this way, nodes that did not seem promising at first can be revisited if the result from expanding an initially more promising node turns out to be less effective than what was anticipated at first [8].

4.3 Adapt the theory to the system

In this section, the framework presented in section 4.2 is applied on the system represented by the model in section 4.1 and the demands on flexibility in the scheduling according to the system specifications in chapter 2.

Firstly, a method is decided. To maximize the efficiency of the system, a job schedule should optimally be used. The real time events affecting the system, such as deactivated and activated machines, jobs that failed a test, etc. are considered as both common and unpredictable. Therefore, the schedule generation should consist of a nominal schedule. Since a single machine, the robot, is in the central of the system in a way that all jobs have to go through this machine when moving to another, a real time event in one part of the system will affect the entire system. The real time events also impose new system restrictions and goals, for example more available machines or a job being cancelled, meaning the real time events cannot be taken care of by right-shifting the schedule. In conclusion, the schedule repair should be based on complete rescheduling.

Secondly, a strategy is decided. As previously stated, some real time events cannot be predicted, however, there are also some predictions that can be made, namely predicting or estimating the processing times within the system. Taking this into account, the most appropriate strategy is to use predictive reactive scheduling.

Thirdly, a policy is decided. The real time events affecting the system, as previously stated, change the direct restrictions and goals of the system. These events need to be considered as soon as they occur. Therefore, event driven rescheduling should be used.

Finally, the alternative approaches that were studied in subsection 4.2.1, 4.2.2 and 4.2.3. To limit the scope of the project, it was chosen to focus on heuristic rules, the branch and bound method, A* and beam search in the implementation. These methods represent all three approach categories and furthermore, the branch and bound method and A* have potential to be less time consuming than dynamic programming and beam search builds on a combination of branch and bound, or A*, and heuristic rules which lower the development time compared to if genetic algorithms or tabu search were to be implemented instead.

Firstly, the method of using heuristic rules is adapted. This method does not generate a schedule, instead the actions are decided in real time. As stated in section 4.1 the jobs have no defined due date and the system does not know about jobs that have not yet arrived to the system. Furthermore, it should be possible to set different priorities for different jobs. The main objective is to maximize the throughput while no jobs get stuck in the system. As stated in subsection 4.2.1, the shortest setup time first rule and the least flexible job first rule tend to maximize the throughput of the system while the earliest release date first rule tends to minimize the variation in waiting time. In this case, there is a considerable chance that the jobs within the system have an equal amount of flexibility due to the possibility of adding more machines to the system to adapt according to the flow of production. A combination of the shortest setup time first rule, the earliest release date first rule and prioritization weights therefore have potential to be effective in this case. However, as stated in subsection 4.2.1, no guarantees for an optimal solution can be given. It is not a goal to minimize the variation in waiting time but the variation should be limited to a certain level.

The combined heuristic rules should give a higher priority for jobs with a lower setup time and it should be possible to assign jobs with prioritization weights. The structure of the system, see figure 4.1, shows that each move consists of going through M_0 to the next location, which can be another machine. Therefore, the setup time to consider is the setup time of M_0 plus any remaining setup time for the upcoming machine in case the setup has not finished once the job arrives at the machine. The processing time for the test stations only depend on the jobs and can therefore not be altered. However, the processing time for the robot, M_0, depends on the chosen schedule. Therefore, a lower processing time in M_0 should increase the priority. To give priority to jobs that have been in the system substantially longer than other jobs, the priority should increase exponentially with increased waiting time, i.e. as the release date gets older. The suggested prioritization function for each job is shown in equation 4.1 and visualized in figure 4.5. The structure of this equation was inspired by the apparent tardiness heuristic, which is a combination of the weighted shortest processing time first rule and the minimum slack first rule, suggested by Pinedo as a solution for a system with one machine and several jobs [5].

$$I_j(t) = \frac{w_j}{a_p p_j + a_s s_j} e^{a_r(t-r_j)}, \quad t \geqslant r_j \tag{4.1}$$

$$w_j = Prioritization\ weight,\ job\ j$$
$$p_j = Processing\ time,\ job\ j$$
$$s_j = Setup\ time,\ job\ j$$
$$r_j = Release\ date,\ job\ j$$
$$a_p = Scaling\ parameter,\ processing\ time$$
$$a_s = Scaling\ parameter,\ setup\ time$$
$$a_r = Scaling\ parameter,\ release\ date$$
$$t = Time$$

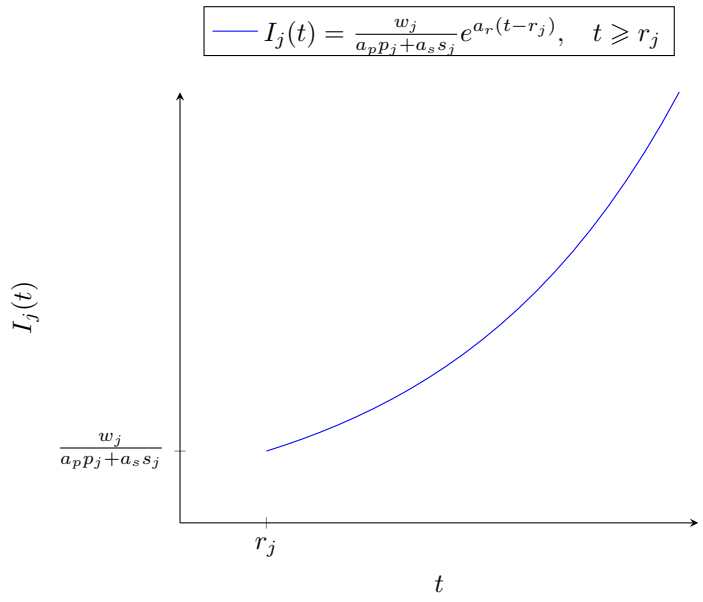

Figure 4.5: The combined heuristic rules as a function of time.

Secondly, the theory of the branch and bound method and A* is adapted. In job scheduling, the goal is not to find an optimal route between two nodes, but to find an optimal sequence for processing all of the jobs. The nodes, in subsection 4.2.2 formulated as positions in the map, can be reformulated into a representation of the system that describes which jobs have been processed in which machines and where in the system the jobs are currently located. The cost of each action, previously formulated as a distance between nodes, can be formulated as the sum of the remaining processing time for a certain job in a certain machine and the proportion of the setup time that can not be finished before the job is ready to enter the machine. The processing times of each test station only depend on what type of job that is being processed. The processing time of the robot depends on the position of the job and the position of the next test station for that job. The setup time of the robot depends on the current position of the robot and the position of the job it should pick up. The setup time of the test station not only depends on the job that is going to be processed, but also the previously processed job. Since the test cycle defines which type of test stations each job should go through and the setup times for the test stations and the robot as well as the processing time of the robot depends on the generated schedule, the processing times of the remaining test stations for a job plus a lower bound of the processing times in the robot can be used as a lower bound until a job is finished. Since jobs can be processed in parallel, the job with the highest lower bound gives the lower bound for finishing all jobs within the system.

This technique would find an optimal solution for processing all jobs the quickest. However, prioritization on different jobs are not taken into account. To take this into account, the representation of the cost would have to be reformulated to a function dependent on not only time but also the prioritization number of the involved jobs.

Finally, combining the branch and bound method or A* with the derived heuristic rule form the beam search method. Taking the time complexity of evaluating the heuristic rule and the limited amount of nodes at each level into account, it was chosen not to split up the filtering into two stages. The beam width should be variable in the simulations.

Chapter 5

Simulations

Before the implementation of the simulation program could start, the purpose of it had to be established. It was discussed whether the focus should be on fully mimicking the database concept to get a complete proof of concept where the user can observe and manipulate the system in the middle of a running simulation or if the focus should be on simulating larger production sequences where utilization factors and production times can be simulated in a predefined system setup. Out of these two alternatives, the latter was more in line with the needs of the company. Axis requested that the simulation program should be written in C#. The student did not have previous experiences from C# so the workflow started of with an introduction to the language, focusing on general syntax, multi-threading and user interfaces. This chapter describes the structure and capabilities of the program. Due to the size of the program, over 3900 lines of code with relatively long algorithms, the complete code will not be presented in this report. Instead, the focus will be on describing the structure and algorithms in text with the help of shorter code snippets.

5.1 General structure

Each of the different subsystems visualized in figure 3.1 plus the scheduler and secondary database, visualized in figure 3.2, were implemented as separate threads. A class that represents jobs were created and to simulate jobs moving within the system, objects of the job class are sent between the different threads using channels were each channel has a reader and a writer. For example, the assembly thread has a channel writer that sends job to the channel reader in the pretest thread. When the movement is initialized from the receiving thread, i.e. when a job should be picked by the robot, the robot thread sends a request to a subsystem which in turn sends a job as response. For example, the robot sends requests containing a position index to the test stations thread using one channel. The test stations thread then sends the job currently located in that position to the robot thread through another channel.

The communication between the threads was implemented in the same way as the movement of jobs, i.e. by sending the information in channels. To synchronize the timing between the different threads, the scheduler sends out updated time stamps to each time dependent thread throughout the course of the simulation.

5.2 Configuration

When starting the simulation, the user is presented with a graphical user interface made with windows forms [4]. The user is requested to select a configuration file for the simulation. The simulation uses configuration files written in the file format TOML, Tom's Obvious Minimal Language, [9] and an example of a configuration file can be found in listing 5.1.

As seen in listing 5.1, the user can specify a failure rate of the pretests, that will be sent from the main thread to the pretest thread, and breakdown times for the 2 positions in the arrivals area, which will be sent to the arrivals thread. In this example the breakdown times are set to -1 and 3000, this represents that the first position will not break but the second position will break 3000 seconds into the simulation.

Continuing reading the file, the user can select the parameters for the heuristic rule from equation 4.1 which will be sent to the scheduler thread.

Furthermore, the user defines the jobs that are going to be sent from the assembly area during the simulation. Each defined job configuration represents an incoming rate of a certain type of jobs with a specified test cycle and prioritization value. In this example 50 jobs of type 1 with prioritization 3 will be sent with 180 seconds between each job starting at time 0. The test cycle consists of test stations of type 1, 2 and 3 where type 1 should go first but the order between type 2 and 3 does not matter.

Further down in the example, the buffer is configured. The size of the buffer is decided and, similarly to the arrivals area, a breakdown time can be set for each position in the buffer. Each position in the buffer can also be given an initial job, a job that is located in that buffer position when the simulations start. The buffer configuration is not only sent to the buffer thread, the scheduler will also receive the size of the buffer and the main thread will, during the configuration, send the prioritization value and test cycle of any jobs initially located in the buffer to the database thread for storage.

Finally, the example shows how test stations are configured. Test stations of specified types are entered one after another in an order that corresponds to the position they are located at. Similarly to the buffer and arrivals area, each test station has a breakdown time but also a connect time, representing when the test station is connected to the system and, from there on, available for usage. Furthermore, each test station is given an initial job configuration and, potentially, a job that is located within the test station when it is connected to the system.

```
PretestFailureRate = 0.03

ArrivalsBreakdownTimes = [-1.0, 3000.0]

[SchedulingConfig]
HeuristicRuleProcessTimeScaling = 1.0
HeuristicRuleSetupTimeScaling = 1.0
HeuristicRuleReleaseDateScaling = 0.05

[[AssemblyJobConfigs]]
JobType = 1
TestCycleTypes = [1, 2, 3]
TestCycleOrderSpecification = [true, false, false]
NbrOfJobs = 50
FirstSendTime = 0.0
TimeBetweenSends = 180.0
Prio = 3

[[BufferConfigs]]
BreakdownTime = -1.0
[BufferConfigs.InitialJob]
JobType = 2
TestCycleTypes = [1, 3]
TestCycleOrderSpecification = [true, true]
Prio = 2

[[BufferConfigs]]
BreakdownTime = 3000.0
[BufferConfigs.InitialJob]

[[TestStationConfigs]]
Type = 1
ConnectTime = 0.0
BreakdownTime = 2500.0
TestFailureRate = 0.01
InitialJobTypeConfig = 1
[TestStationConfigs.InitialJob]

[[TestStationConfigs]]
Type = 2
ConnectTime = 0.0
BreakdownTime = -1.0
TestFailureRate = 0.01
InitialJobTypeConfig = 1
[TestStationConfigs.InitialJob]
JobType = 3
TestCycleTypes = [1, 3, 2]
TestCycleOrderSpecification = [false, false, true]
Prio = 2

[[TestStationConfigs]]
Type = 3
ConnectTime = 0.0
BreakdownTime = -1.0
TestFailureRate = 0.01
InitialJobTypeConfig = 1
[TestStationConfigs.InitialJob]

[[TestStationConfigs]]
Type = 1
ConnectTime = 3000.0
BreakdownTime = -1
TestFailureRate = 0.01
InitialJobTypeConfig = 2
[TestStationConfigs.InitialJob]
```

Listing 5.1: An example of a configuration file.

5.3 The assembly thread

The assembly thread receives the job configuration and sends the time when the first jobs are going to be sent from the assembly area together with the amount of jobs that are sent at that time to the scheduler thread. Thereafter, the assembly area waits for an update of the time stamp from the scheduler. When the time stamp matches the time when the next job is going to be sent, the assembly area gives that job an identification number, sends the prioritization value and test cycle to the database thread and thereafter sends the job to the pretest area. After this is done, the assembly area will also send the time when the next jobs are sent together with the number of jobs sent at that time to the scheduler. The scheduler needs this information to know how the time stamp should be updated.

5.4 The pretest thread

Every time a job is received to the pretest thread, the pretest thread will perform a simulated test where the chance of passing is in accordance with the failure rate configuration. The job will only be sent forward to the arrivals area if it passes the pretest. Otherwise, the job is discarded. The time of a pretest is set to 0, meaning that the job is sent forward at the same time stamp as it arrived to the area. The pretest thread will also send a notification to the scheduler, telling whether each carried out pretest is passed or not. This information tells the scheduler whether it should expect an arriving job or not.

5.5 The arrivals thread

When the arrivals thread receives its configuration, the potential times of breakdowns are sent to the scheduler to be used when determining the update of the time stamp. The arrivals thread receives the jobs from the pretest thread and notifies the scheduler of the arrival. This notification consists of the position of the job, its type number and its position in the arrivals area. If there are more than two jobs in the arrivals area, jobs will be placed in a queue and the notification for that job will not be sent until the job has reached one of the two positions that are reachable by the robot. If a position in the arrivals area is set to experience a breakdown, the thread will notify the scheduler about the breakdown at the time it occurs. Jobs will still arrive to the area but the scheduler will not be notified about these jobs, meaning that if a job arrives to that position, it will be stuck there throughout the rest of the simulation, which will be accounted for by the scheduler.

5.6 The robot thread

The robot thread receives commands from the scheduler. The commands tell the robot to pick a job from a certain position or to place a job at a certain position. When the action is completed, the robot will notify the scheduler. The robot itself does not handle any time durations for the movements, however the scheduler uses a function for calculating the time durations for each action of the robot.

5.7 The buffer thread

When the buffer has received its configuration, it will notify the scheduler of any initial jobs in the buffer and off any upcoming breakdown times. Similarly to the arrivals thread, the buffer thread will also notify the scheduler when a predefined breakdown occurs at a specific position within the buffer, any jobs currently located at that position will be discarded. Whenever the buffer receives a job from the robot, it will notify the scheduler as a confirmation that the correct job has arrived. This is, however, not utilized in the scheduler but in a future update it could be used as another way of inserting simulated errors in the system.

5.8 The test stations thread

A single thread handles all the test stations. When this thread receives its configuration, the connect times and breakdown times are forwarded to the scheduler. The test times for each job type in each test station are stored in the database. When the test stations thread receives its configuration, it will also request the test times for all types of test stations included in the configuration from the database thread. When a test station is connected, the scheduler will be informed of the type of the test station, the position, the configuration and the id and type of any initial job in the test station. Similarly to the buffer and arrivals thread, the scheduler will also be notified when a test station is experiencing a predefined error. Before each test, the test stations thread receives information on the upcoming test from the scheduler, the test station in question will then prepare by changing the configuration to match the upcoming job type. When the job is received, the test stations thread will calculate when the test should be finished. When the time stamp matches this time, the test stations thread notifies the scheduler with a test result that is randomized according to the configured failure rate.

5.9 The scheduler thread

A simplified pseudo code version of the main loop of the scheduler is shown in listing 5.2. This loop is entered once the scheduler has received a complete configuration. First, the initial jobs must become available. The scheduler will check when the first event occurs, i.e. arriving jobs, connected test stations, or breakdowns, etc. The time stamp is then updated accordingly. Then the scheduler will check if there will occur any breakdowns at this time stamp. If so, the scheduler will wait for a breakdown notification. Next up, the scheduler will check for arriving jobs, connected test stations and any initial jobs in a test station or in the buffer. For any arriving jobs, the scheduler will request the prioritization value and test cycle from the database. If a new job type arrives for the first time, the scheduler will also request the test times for that type of job from the database. All this is done in line 1-6 in listing 5.2. It is enclosed in a while loop since pretest failures and breakdowns can stop expected jobs from becoming available.

Then the main while loop is entered and will not be exited until the scheduling is done or has failed due to a deadlock. The scheduler will first use the information of all available jobs and determine the best action to take using the heuristic rule defined by equation 4.1. This function will be explained in more detail later in this section. If no action is available, either a deadlock has been achieved or, more commonly, the scheduler needs to wait for tests to finish. Therefore, if no action is available, the scheduler will perform the steps from line 3-5 again. What was not mentioned before, since it is not relevant until the first test has started, is that when checking for available jobs, test results are also received. If, however, an action is available the scheduler will command the robot to perform this action.

If the end position is a test station the scheduler will start by sending the information of the job to the receiving test station. Then the scheduler will send the pick and place commands to the robot while also updating the time stamp with the time duration of the action. During this procedure, new events are also registered, again similarly to line 3-5. In cases where the end position is a test station, once the action has been performed, the time when the test is finished is stored. To calculate this time, the test time is added to any remaining time duration for changing job configuration in the test station, in case the time duration for changing configuration is longer than the time duration of the action. The job will be marked as unavailable until a test result is received. If the end position on the other hand is the buffer, the job will remain available and if the end position is the passed or not passed area, the job will be removed.

Once the action has been performed, the scheduler will once again update the time stamp and receive the new system state. The loop then continues until there are no more jobs to handle or until a deadlock has occurred. Once at this point, the scheduler will send the results of the simulation to the main thread.

```
while (No jobs available && More jobs arriving)
{
    UpdateTimeStampToNextEventTime();
    CheckForBreakdowns();
    ReceiveAvailableJobsNTs();
}

while ((Jobs available || Jobs being tested) && No deadlock)
{
    (bestIdFp, bestStartPos,
    bestEndPos, bestSetupTimeRobot,
    bestProcessTimeRobot,
    bestRemainingsetupTimeTs) = GetNextActionHeuristicRule();

    if(bestStartPos == null)
    {
      UpdateTimeStampToNextEventTime();
      CheckForBreakdowns();
      ReceiveAvailableJobsNTs();
    }
    else
    {
      PerformAction(bestStartPos, bestEndPos,
                    bestIdFp, bestSetupTimeRobot,
                    bestProcessTimeRobot,
                    bestRemainingsetupTimeTs);
  }

  while (No jobs available && More jobs arriving)
  {
      UpdateTimeStampToNextEventTime();
      CheckForBreakdowns();
      ReceiveAvailableJobsNTs();
  }
}
SendResult();
```

Listing 5.2: The main loop of the scheduler thread.

The algorithm for selecting the next action using the heuristic rule from equation 4.1 will now be explained. The complete algorithm can be found in appendix A and a visualization of the algorithm can be found in 5.1. The algorithm checks the available actions for each available job. Each job can either have failed a test or be done with its test cycle, have only one type of test station as an option for the next action, or have several different types of test stations as the next option.

In the first case, if the job has either failed a test or is done with its test cycle, see line 15-49 in appendix A or the upper left branch in figure 5.1, the job should be moved to the passed or not passed area. The algorithm calculates the setup time of the robot, i.e. the time to get to the job, and the process time of the robot, i.e. the time it takes to move the robot to its end area. With the help of these values as well as the prioritization value and release time of the job and the parameters defined in the configuration, a scheduling prioritization value is calculated using equation 4.1. If this is the highest value found so far, the action will be saved as the best one so far.

Continuing with the second case, when the job only have one type of test station to choose from, see line 50-124 in appendix A or the upper middle branch in figure 5.1. The algorithm loops through each of the connected test stations and checks if they are of the right type. For test stations of the correct type, the algorithm checks if the test station is occupied by another job that is done with its test. If this is true and no available action can be found, that is the sign of a deadlock. Therefore, the test station in question will be stored together with the job requesting to go there so that the information can be used if a deadlock has occurred. Otherwise, if the test station is not occupied and it has not been reserved by another job to solve a deadlock, this is explained further down in the algorithm, the scheduling prioritization value is calculated in the same way as in the first case, with one exception. If the job is already located at a test station of the correct type, the job will receive the highest possible scheduling prioritization value and the algorithm will return the action. This is because such an action would take zero time and no other action that requires manipulation by the robot can achieve a higher scheduling priority.

Finally the third case, when the test cycle of the job allows for several different types of test stations next, see line 125-223 in appendix A or the upper right branch in figure 5.1. Here, the algorithm first checks how many of the upcoming tests in the test cycle can be performed in any order. Then the algorithm will run in the same way as in the second case, except that it will run through that algorithm for all different types of test stations available to chose from.

At this point, all actions have been evaluated and the best action can be returned. However, if no action has been found and deadlock situations have been noticed during the algorithm, the deadlock has to be solved using the buffer, see line 224- 309 in appendix A or the lower branch in figure 5.1. The algorithm will loop through all buffer positions and store the buffer positions that are currently available. If no positions are available and there are no jobs that are currently undertaking tests, that means that the deadlock cannot be solved. The function will then return and the scheduling has failed. If that is not the case, however, the algorithm will loop through all deadlock situations that were stored earlier in the algorithm. The stored information is the position of a job and the position of the test station it seeks. For each deadlock situation, the scheduler will check if the job in the sought test station seeks the test station that the currently checked job is located at, i.e. if two jobs want to switch position, here called a double sided deadlock. Then the algorithm will loop over all available positions in the buffer and calculate the scheduling priority. However, double sided deadlocks will be prioritized over non double sided deadlocks, since solving such a deadlock would solve two deadlocks at the same time and thereby increase the chance of being able to finish the scheduling. When the scheduling priority is calculated, moving a job to the buffer counts as favourable for the job that seeks the test station the moved job was located in. Therefore, the prioritization value and release time of the job seeking the test station is used in the formula, equation 4.1, instead of the values for the job that is actually being moved. To make sure that the system does not go back into a similar deadlock after the next action, for example by letting a job from the arrivals area be moved to the test station in question, the two jobs involved in the deadlock that scored the highest scheduling priority will have the test station types they seek reserved until they have been moved there.

In the simulation, this algorithm also includes collecting of statistical data. This is however excluded from the versions presented in appendix A and figure 5.1 to simplify the understanding of the algorithm. When looking through the algorithm it was noticed that the algorithm checks for a double sided deadlock for every available buffer position, which is less effective compare to only checking once before looping through the buffer positions. This was fixed in the visualization in figure 5.1 but since the project had only been handed in at this time it was not fixed in appendix A, to better reflect the implemented code. However, to fix this, the code on line 169 should switch place with the code on line 271 in appendix A, while keeping the initial indentation.

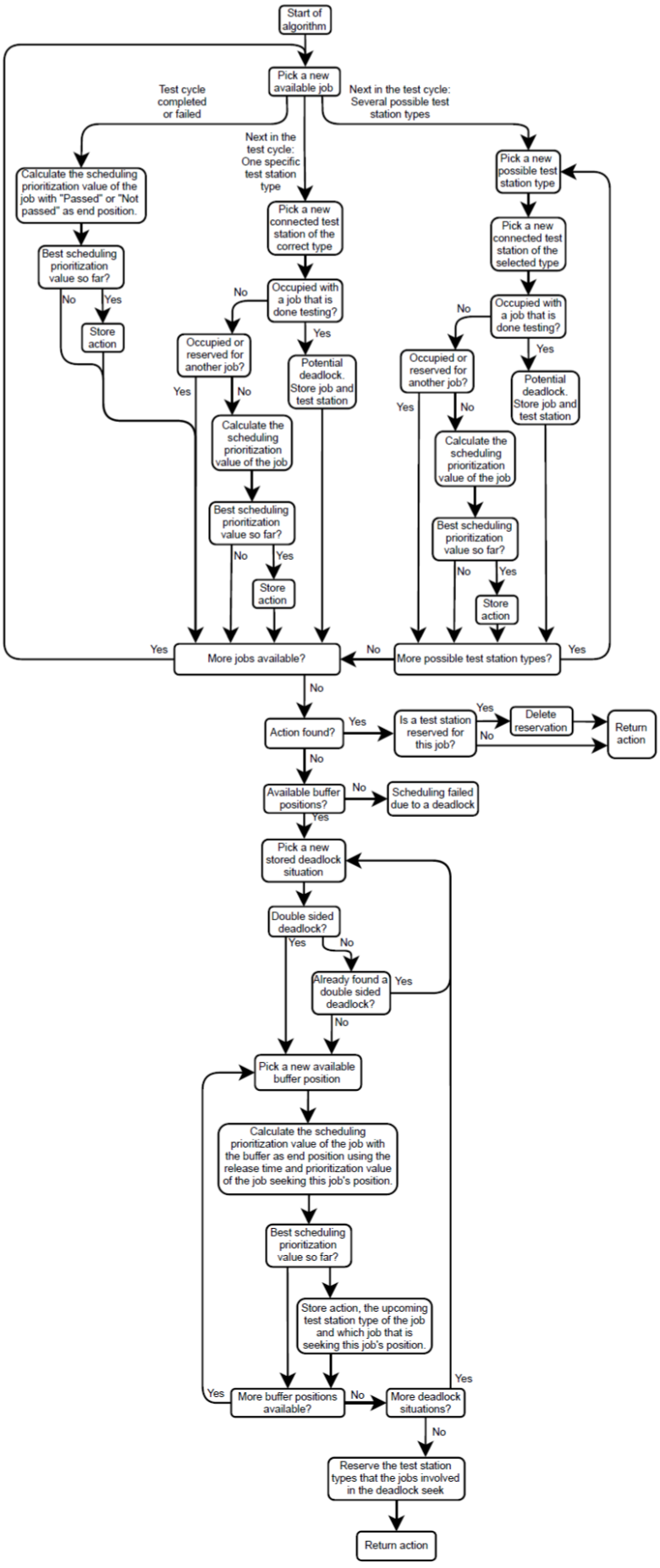

Figure 5.1: A visualization of the scheduling algorithm.

5.10 Presenting the result

When the scheduling is finished or has failed, the result is sent to the main thread. The result consist of the following data. If the scheduling was successful or not, the total time of the production sequence, the throughput, the total number of jobs and the number of jobs that passed, failed a test, failed a pretest, and got stuck. Furthermore, the result contains the utilization factors of the robot and each of the test stations. For the robot, the utilization factor is the percentage of the total time it spent moving jobs. For the test stations, the utilization factor is the percentage of the time when the test station was active, i.e. connected and not broken, that was spent performing tests. For each test station, the result also includes the percentage of the time when the test station was active that was spent waiting for job configuration changes to finish while already having a job available for testing. When it comes to the jobs, for each type of job, the result stores the average percentage of the time from the moment when the job reached a reachable position in the arrivals area, or equivalent for initial jobs in the buffer or in a test station, to the moment it left the system or got stuck that was spent undertaking tests, waiting for not being prioritized by the scheduler, waiting for the robot to arrive at the position of the job, being moved by the robot and waiting for test station types to become available. Note that these time percentages does not add up to one since jobs can be waiting for several different types of test stations as well as not being prioritized by the scheduler at the same time. Furthermore, the time from when the job becomes available, in the arrivals area or after a finished test, until the robot is ready for the next action is not included in the statistics. The result also stores utilization data concerning the buffer. The data that is stored shows the percentage of the total time that the buffer stored zero jobs, one job, etc., up to the size of the buffer, excluding any jobs in broken buffer positions. This data is shown in the graphical user interface, as seen in figure 5.2, and the user has the option to save it as a TOML-file together with the used configuration.

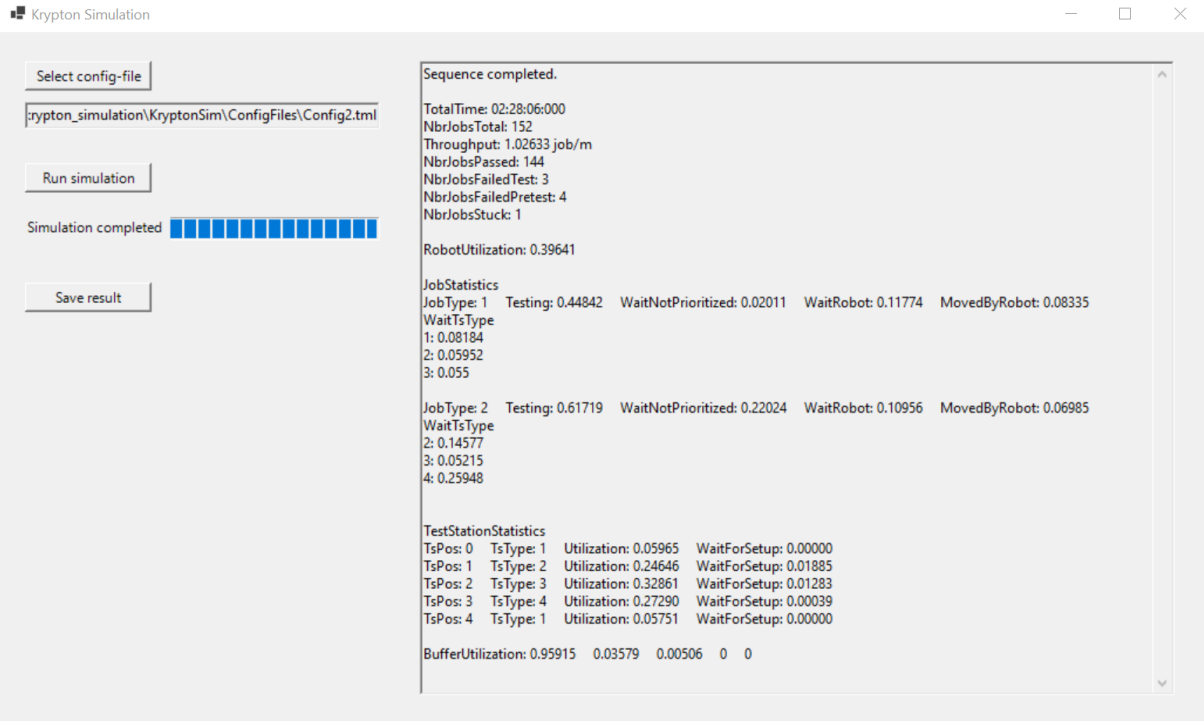

Figure 5.2: An example of the graphical user interface after running a production sequence.

Several simulations with different configurations were run to verify that the program had the desired behavior. However, because of the inaccuracy in the estimated test times and movement times as well as the arbitrary test cycles selected, no general conclusion regarding performance can be said. Instead, the developed program is meant to be used as a tool to assess what the necessary system layout would be for different production rates.

Chapter 6

Conclusions and discussion

This project started off by taking the vision of a future system and formulating it into a concrete system specification, a first step in bringing the vision to reality. The system specifications were analysed to create two concepts for the flow of information within the system. While the two concepts offer similar capabilities, the database concept was recommended for a future implementation thanks to the high experience and knowledge of similar systems within the project group. The solution provides a flexible system where test stations can be connected and disconnected during production, a stable system where both jobs that are moved in an unexpected way, for example by an operator, and subsystems that are experiencing errors are noticed and accounted for automatically, all this in accordance with the goals stated in chapter 1.

The theories behind job shop scheduling were studied in chapter 4 to be able to maximize the efficiency of the system, again in accordance with the goals stated in chapter 1. The heuristic rule that was derived, see equation 4.1, focuses, as stated in section 4.3, on maximizing the throughput. However, as also stated, heuristic rules leave no guarantees and parameters have to be tweaked until a satisfying result is achieved. The technique based on classical optimization, A*, guarantees an optimal solution but this algorithm requires substantially higher processing power compared to using a heuristic rule. When designing the system one has to weigh the need for optimality against the increased hardware cost it would entail. As also stated in chapter 4, beam search can be used as a compromise between these two techniques.

Since the implemented simulation is based on the database-concept, it can be used as a proof of concept. Furthermore, the simulations can be used to assess how many test stations that are needed to achieve the requested production rate and how large the buffer should be to be able to avoid deadlocks when running different test cycles. However, to do this analysis the estimated test times and robot movement times would have to be updated to more precise approximations. It is also recommended that the project team continues the development by including A* and beam search as alternatives along side the heuristic rule. With all three approaches implemented, the performance and processing requirements can be compared before a technique is chosen for a future implementation.

Once the simulations have given a clear assessment of the required number of test stations etc., the database concept can be used as a base when completing a full design of the physical system upon its implementation.

This has been an exciting and educational project to work on. I hope and believe that the result will be of great help when implementing the system and that the future implementation will help Axis increase their productivity.

Bibliography

[1] *Collimator*. Encyclopaedia Britannica. June 25, 2012. URL: https://www.britannica.com/technology/collimator (visited on 12/17/2020).

[2] Axis Communications. *About Axis*. URL: https://www.axis.com/sv-se/about-axis (visited on 08/22/2020).

[3] Shawn Frayne. *All about Proximity Sensors: Which type to use?* Seeed Studio. 2019. URL: https://www.seeedstudio.com/blog/2019/12/19/all-about-proximity-sensors-which-type-to-use/ (visited on 12/11/2020).

[4] Andy De George. *Desktop Guide (Windows Forms .NET)*. Microsoft. Oct. 26, 2020. URL: https://docs.microsoft.com/en-us/dotnet/desktop/winforms/overview/?view=netdesktop-5.0 (visited on 03/13/2020).

[5] Michael L. Pinedo. *Planning and Scheduling in Manufacturing and Services*. 2nd ed. New York, NY, USA: Springer New York, 2009, pp. 441–445. ISBN: 978-1-4419-0909-1.

[6] Michael L. Pinedo. *Planning and Scheduling in Manufacturing and Services*. 2nd ed. New York, NY, USA: Springer New York, 2009, pp. 423–427. ISBN: 978-1-4419-0909-1.

[7] Michael L. Pinedo. *Planning and Scheduling in Manufacturing and Services*. 2nd ed. New York, NY, USA: Springer New York, 2009, pp. 430–432. ISBN: 978-1-4419-0909-1.

[8] Michael L. Pinedo. *Planning and Scheduling in Manufacturing and Services*. 2nd ed. New York, NY, USA: Springer New York, 2009, pp. 449–461. ISBN: 978-1-4419-0909-1.

[9] Tom Preston-Werner. *TOML; Tom's Obvious Minimal Language*. URL: https://toml.io/en/ (visited on 03/13/2021).

[10] Christian Rosen and Gustaf Olsson. *Industrial automation*. Rev. ed. Lund, Sweden: Media-Tryck, Lund University, 2005, pp. 512–520.

[11] Peter Norvig Stuart Russell. *Artificial intelligence: A modern approach*. 3rd ed. Harlow, England: Pearson Education Limited, 2016, pp. 98–99. ISBN: 978-1-1292-1539-64.

[12] Zhen Wang, Jihui Zhang, and Jianfei Si. "Dynamic Job Shop Scheduling Problem with New Job Arrivals: A Survey". In: *Proceedings of 2019 Chinese Intelligent Automation Conference*. Ed. by Zhidong Deng. Singapore: Springer Singapore, 2020, pp. 664–668. ISBN: 978-981-32-9050-1.

[13] Eric W. Weisstein. *NP-Problem*. MathWorld -A Wolfram Web Resource. URL: https://mathworld.wolfram.com/NP-Problem.html (visited on 11/15/2020).

[14] Patrick Winston. *MIT 6.034 Artificial Intelligence, Lecture 5: Search: Optimal, Branch and Bound, A**. Massachusetts Institute of Technology. Cambridge, MA, USA. 2010. URL: https://ocw.mit.edu/courses/electrical-engineering-and-computer-science/6-034-artificial-intelligence-fall-2010/lecture-videos/ (visited on 11/17/2020).

Appendices

Appendix A: The scheduling algorithm to get the next action

```
public (int, Position, Position, double, double, double) GetNextActionHeuristicRule()
{
  int bestJobId = 0;
  Position bestStartPos = null;
  Position bestEndPos = null;
  double bestSetupTimeRobot = 0;
  double bestProcessTimeRobot = 0;
  double bestRemainingsetupTimeTs = 0;
  double bestSchedulingPrio = 0;

  List<Position[]> jobPosToOccupiedTsPos = new List<Position[]>();

  foreach (JobInfoScheduling jobInfo in this.AvailableJobs)
  {
    if (jobInfo.FailedTest ||
        jobInfo.PerformedTests.Count == jobInfo.TestCycleTsTypes.Count ||
        jobInfo.PerformedTests.Count == jobInfo.TestCycleTsTypes.FindIndex(type =>
        type == 0))
    {
      double setupTimeRobot = Globals.GetRobotMoveTime(this.RobotPos, jobInfo.Pos);
      Globals.Areas endArea;
      if (!jobInfo.FailedTest)
      {
        endArea = Globals.Areas.Passed;
      }
      else
      {
        endArea = Globals.Areas.NotPassed;
      }
      double processTimeRobot = Globals.GetRobotMoveTime(jobInfo.Pos, new
      Position(endArea, 0));

      double schedulingPrio = jobInfo.Prio / (this.HeuristicRuleProcessTimeScaling *
                              processTimeRobot +
                              this.HeuristicRuleSetupTimeScaling * setupTimeRobot)
                              * Math.Exp(this.HeuristicRuleReleaseDateScaling *
                              (this.TimeStamp - jobInfo.ReleaseTime));

      if (schedulingPrio > bestSchedulingPrio)
      {
        bestJobId = jobInfo.IdFp;
        bestStartPos = jobInfo.Pos;
        bestEndPos = new Position(endArea, 0);
        bestSetupTimeRobot = setupTimeRobot;
        bestProcessTimeRobot = processTimeRobot;
        bestRemainingsetupTimeTs = 0;
        bestSchedulingPrio = schedulingPrio;
      }
    }
    else if (jobInfo.TestCycleOrders[jobInfo.PerformedTests.Count])
    {
      foreach (TestStationInfoScheduling tsInfo in this.TestStations)
      {
        if (tsInfo.Type == jobInfo.TestCycleTsTypes[jobInfo.PerformedTests.Count])
        {

```

```
57          if (tsInfo.Occupied && jobInfo.Pos.Area == Globals.Areas.TestStations
58              && jobInfo.Pos.Index != tsInfo.IndexPos
59              && this.AvailableJobs.Any(x => (x.Pos.Area ==
60                Globals.Areas.TestStations && x.Pos.Index == tsInfo.IndexPos)))
61          {
62            jobPosToOccupiedTsPos.Add(new Position[] { jobInfo.Pos, new
63            Position(Globals.Areas.TestStations, tsInfo.IndexPos) });
64          }
65          else if ((!tsInfo.Occupied || (jobInfo.Pos.Area ==
66                  Globals.Areas.TestStations && jobInfo.Pos.Index == tsInfo.IndexPos))
67                  &&(!this.AntiDeadlockTsTypeRecervedJobId.Any(x => x[0] == tsInfo.Type)
68                  || this.AntiDeadlockTsTypeRecervedJobId.Where(x => x[0] ==
69                  tsInfo.Type).Any(x => x[1] == jobInfo.IdFp)))
70          {
71            double schedulingPrio;
72            double processTimeRobot;
73            double setupTimeRobot;
74            double remainingsetupTimeTs;
75
76            double setupTimeTs = Globals.GetTsSetupTime(tsInfo.JobTypeConfig,
77            jobInfo.Type);
78
79              if (jobInfo.Pos.Area == Globals.Areas.TestStations && jobInfo.Pos.Index ==
80              tsInfo.IndexPos)
81                {
82              schedulingPrio = double.MaxValue;
83              processTimeRobot = 0;
84              setupTimeRobot = 0;
85              remainingsetupTimeTs = setupTimeTs;
86            }
87            else
88            {
89              processTimeRobot = Globals.GetRobotMoveTime(jobInfo.Pos, new
90              Position(Globals.Areas.TestStations, tsInfo.IndexPos));
91              setupTimeRobot = Globals.GetRobotMoveTime(this.RobotPos, jobInfo.Pos);
92
93              remainingsetupTimeTs = Math.Max(setupTimeTs - (setupTimeRobot +
94              processTimeRobot), 0);
95
96              schedulingPrio = jobInfo.Prio / (this.HeuristicRuleProcessTimeScaling *
97                              processTimeRobot +
98                              this.HeuristicRuleSetupTimeScaling *
99                              (setupTimeRobot + remainingsetupTimeTs))
100                             * Math.Exp(this.HeuristicRuleReleaseDateScaling
101                             * (this.TimeStamp - jobInfo.ReleaseTime));
102           }
103
104           if (schedulingPrio > bestSchedulingPrio)
105           {
106             bestJobId = jobInfo.IdFp;
107             bestStartPos = jobInfo.Pos;
108             bestEndPos = new Position(Globals.Areas.TestStations, tsInfo.IndexPos);
109             bestProcessTimeRobot = processTimeRobot;
110             bestSetupTimeRobot = setupTimeRobot;
111
112             bestRemainingsetupTimeTs = remainingsetupTimeTs;
113             bestSchedulingPrio = schedulingPrio;
114
115             if(bestSchedulingPrio == double.MaxValue)
116             {
117               return (bestJobId, bestStartPos, bestEndPos, bestSetupTimeRobot,
118               bestProcessTimeRobot, bestRemainingsetupTimeTs);
119             }
120           }
121         }
122       }
123     }
124   }
125   else
126   {
127     int i = jobInfo.PerformedTests.Count;
128
129
```

```
130       while ((i < jobInfo.TestCycleTsTypes.Count ||
131           i < jobInfo.TestCycleTsTypes.FindIndex(type => type == 0))
132           && !jobInfo.TestCycleOrders[i])
133       {
134         i++;
135       }
136
137       List<int> testCycleTypesSublist = jobInfo.TestCycleTsTypes.GetRange(0, i);
138       i -= 1;
139       while (i >= 0 && !jobInfo.TestCycleOrders[i])
140       {
141         foreach (TestStationInfoScheduling tsInfo in this.TestStations)
142         {
143           if (tsInfo.Type == jobInfo.TestCycleTsTypes[i]
144               && jobInfo.PerformedTests.Count(x => x.Equals(tsInfo.Type)) <
145               testCycleTypesSublist.Count(x => x.Equals(tsInfo.Type)))
146           {
147             if (tsInfo.Occupied && jobInfo.Pos.Area == Globals.Areas.TestStations &&
                    jobInfo.Pos.Index != tsInfo.IndexPos
148               && this.AvailableJobs.Any(x => (x.Pos.Area == Globals.Areas.TestStations
149                                               && x.Pos.Index == tsInfo.IndexPos)))
150             {
151               jobPosToOccupiedTsPos.Add(new Position[] { jobInfo.Pos, new
152               Position(Globals.Areas.TestStations, tsInfo.IndexPos) });
153             }
154             else if ((!tsInfo.Occupied ||
155                     (jobInfo.Pos.Area == Globals.Areas.TestStations
156                     && jobInfo.Pos.Index == tsInfo.IndexPos))
157                     &&(!this.AntiDeadlockTsTypeRecervedJobId.Any(x => x[0] ==
158                     tsInfo.Type)
159                     || this.AntiDeadlockTsTypeRecervedJobId.Where(x => x[0] ==
160                     tsInfo.Type).Any(x => x[1] == jobInfo.IdFp)))
161             {
162
163               double schedulingPrio;
164               double processTimeRobot;
165               double setupTimeRobot;
166               double remainingsetupTimeTs;
167
168                 double setupTimeTs = Globals.GetTsSetupTime(tsInfo.JobTypeConfig,
169                 jobInfo.Type);
170
171                 if (jobInfo.Pos.Area == Globals.Areas.TestStations
172                     && jobInfo.Pos.Index == tsInfo.IndexPos)
173               {
174                 schedulingPrio = double.MaxValue;
175                 processTimeRobot = 0;
176                 setupTimeRobot = 0;
177                 remainingsetupTimeTs = setupTimeTs;
178               }
179               else
180               {
181                 processTimeRobot = Globals.GetRobotMoveTime(jobInfo.Pos, new
182                 Position(Globals.Areas.TestStations, tsInfo.IndexPos));
183                 setupTimeRobot = Globals.GetRobotMoveTime(this.RobotPos, jobInfo.Pos);
184
185                 remainingsetupTimeTs = Math.Max(setupTimeTs - (setupTimeRobot +
186                                           processTimeRobot), 0);
187
188                 schedulingPrio = jobInfo.Prio / (this.HeuristicRuleProcessTimeScaling
                                  * processTimeRobot
189                                  + this.HeuristicRuleSetupTimeScaling
190                                  * (setupTimeRobot + remainingsetupTimeTs))
191                                  * Math.Exp(this.HeuristicRuleReleaseDateScaling
192                                  * (this.TimeStamp - jobInfo.ReleaseTime));
193               }
194
195
196
197
198
199
200
```

```
              if (schedulingPrio > bestSchedulingPrio)
              {
                bestJobId = jobInfo.IdFp;
                bestStartPos = jobInfo.Pos;
                bestEndPos = new Position(Globals.Areas.TestStations, tsInfo.IndexPos);
                bestProcessTimeRobot = processTimeRobot;
                bestSetupTimeRobot = setupTimeRobot;
                bestRemainingsetupTimeTs = remainingsetupTimeTs;
                bestSchedulingPrio = schedulingPrio;

                if (bestSchedulingPrio == double.MaxValue)
                {
                  return (bestJobId, bestStartPos, bestEndPos, bestSetupTimeRobot,
                  bestProcessTimeRobot, bestRemainingsetupTimeTs);
                }
              }
            }
          }
        }
        i--;
      }
    }
  }
  if (bestSchedulingPrio == 0 && jobPosToOccupiedTsPos.Count > 0)
  {
    List<int> availableBufferPoses = new List<int>();

    for (int i = 0; i < this.BufferSize; i++)
    {
      if (!this.AvailableJobs.Where(x => x.Pos.Area == Globals.Areas.Buffer)
          .Any(x => x.Pos.Index == i)
          && !this.BrokenBuffer[i])
      {
        availableBufferPoses.Add(i);
      }
    }
    if (availableBufferPoses.Count == 0)
    {
      if(this.UnavailableJobs.Count == 0)
      {
        this.Success = false;
      }
    }
    else
    {
      int bestJobIdToTs = 0;
      int bestTsTypeAfterBuffer = 0;
      bool doubleSidedDeadlock = false;
      bool foundDoubleSidedDeadlock = false;
      foreach (Position[] jobNTsPos in jobPosToOccupiedTsPos)
      {
        Position jobPos = jobNTsPos[0];
        Position tsPos = jobNTsPos[1];

        JobInfoScheduling jobToTs = this.AvailableJobs.Find(x => x.Pos == jobPos);
        int tsAfterBuffer = this.TestStations.Find(x => x.IndexPos == tsPos.Index).Type;

        if (jobPosToOccupiedTsPos.Any(x => x[0].Index == tsPos.Index)
            && jobPosToOccupiedTsPos.Find(x => x[0].Index == tsPos.Index)[1].Index
            == jobPos.Index)
        {
          doubleSidedDeadlock = true;
          foundDoubleSidedDeadlock = true;
        }
        else
        {
          doubleSidedDeadlock = false;
        }
        foreach (int bufferIndexPos in availableBufferPoses)
        {
          if(!foundDoubleSidedDeadlock || doubleSidedDeadlock)
          {

```

```
            double processTimeRobot = Globals.GetRobotMoveTime(tsPos, new
            Position(Globals.Areas.Buffer, bufferIndexPos));
            double setupTimeRobot = Globals.GetRobotMoveTime(this.RobotPos, tsPos);

            double schedulingPrio = jobToTs.Prio / (this.HeuristicRuleProcessTimeScaling
                                    * processTimeRobot
                                    + this.HeuristicRuleSetupTimeScaling
                                    * setupTimeRobot)
                                    * Math.Exp(this.HeuristicRuleReleaseDateScaling
                                    * (this.TimeStamp - jobToTs.ReleaseTime));

            if (schedulingPrio > bestSchedulingPrio)
            {
              bestJobId = this.AvailableJobs.Find(x =>
                                          (x.Pos.Area == Globals.Areas.TestStations
                                          && x.Pos.Index == tsPos.Index)).IdFp;
              bestStartPos = tsPos;
              bestEndPos = new Position(Globals.Areas.Buffer, bufferIndexPos);
              bestSetupTimeRobot = setupTimeRobot;
              bestProcessTimeRobot = processTimeRobot;
              bestRemainingsetupTimeTs = 0;
              bestSchedulingPrio = schedulingPrio;


              bestJobIdToTs = jobToTs.IdFp;
              bestTsTypeAfterBuffer = tsAfterBuffer;
            }
          }
        }
      }
      this.AntiDeadlockTsTypeRecervedJobId.Add(new int[] { this.TestStations.Find(x =>
                              x.IndexPos == bestStartPos.Index).Type, bestJobIdToTs });
      this.AntiDeadlockTsTypeRecervedJobId.Add(new int[] { bestTsTypeAfterBuffer,
                                                           bestJobId });
    }
  }
  int indexToRemove = this.AntiDeadlockTsTypeRecervedJobId
                      .FindIndex(x => x[1] == bestJobId);
  if (indexToRemove != -1)
  {
    this.AntiDeadlockTsTypeRecervedJobId.RemoveAt(indexToRemove);
  }
  return (bestJobId, bestStartPos, bestEndPos, bestSetupTimeRobot,
          bestProcessTimeRobot, bestRemainingsetupTimeTs);
}
```