MASTER

Algorithms for parallel machine scheduling with a sequence dependent cost function application in a printed circuit board assembly production environment

Reusken, E.W.

Award date:
2019

Disclaimer
This document contains a student thesis (bachelor's or master's), as authored by a student at Eindhoven University of Technology. Student theses are made available in the TU/e repository upon obtaining the required degree. The grade received is not published on the document as presented in the repository. The required complexity or quality of research of student theses may vary by program, and the required minimum study period may vary in duration.

General rights
Copyright and moral rights for the publications made accessible in the public portal are retained by the authors and/or other copyright owners and it is a condition of accessing publications that users recognise and abide by the legal requirements associated with these rights.

- Users may download and print one copy of any publication from the public portal for the purpose of private study or research.
- You may not further distribute the material or use it for any profit-making activity or commercial gain
Algorithms for parallel machine scheduling with a sequence dependent cost function

Application in a printed circuit board assembly production environment

Master of Science Thesis

For the degree Master of Science in Embedded Systems - Systems on Chip at Eindhoven University of Technology

Erik Reusken
September 4, 2019

Supervisors:
  dr. Kevin Buchin, prof. dr. Frits Spieksma, dr. Daniel Kowalczyk,
  ir. Martijn van de Ven, ir. Leon de Wit

Department of Mathematics and Computer Science, Algorithms Research Group,
Eindhoven University of Technology
Acknowledgment

First, I would like to thank my thesis supervisor dr. Kevin A. Buchin from the Algorithms group at the Department of Mathematics and Computer Science of the Eindhoven University of Technology (TU/e). He brought me in contact with other experts from the algorithmic group, gave valuable input for my research work and steered me in the right direction whenever needed.

I would also like to thank the other experts from the TU/e who were involved in this research project: prof. dr. Frits C.R. Spieksma and dr. Daniel Kowalczyk. In particular their advice concerning limiting the scope, technical and algorithmic possibilities, and suggestions for appropriate mathematical models is highly appreciated.

Furthermore I am grateful to Prodrive Technologies for offering me an internship. They gave me the opportunity and freedom to shape and develop this master thesis project in a direction that suited my interests very well. Due to the project I gained knowledge in different new fields and increased my abstract thinking skills. From Prodrive Technologies, I would like to thank ir. Martijn van de Ven for his innovative suggestions, ir. Leon de Wit for our regular meetings and his support in econometrics and statistics, and ing. Robbert Bax for his suggestions and his support with acquiring all available data.

Finally I thank my friends and family who supported me and kept me motivated during the years of my master studies.
# Table of contents

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ABSTRACT</td>
<td>5</td>
</tr>
<tr>
<td>LIST OF ABBREVIATIONS</td>
<td>6</td>
</tr>
<tr>
<td>LIST OF TERMINOLOGY</td>
<td>8</td>
</tr>
<tr>
<td>1 INTRODUCTION</td>
<td>9</td>
</tr>
<tr>
<td>1.1 BACKGROUND</td>
<td>9</td>
</tr>
<tr>
<td>1.1.1 Prodrive Technologies</td>
<td>9</td>
</tr>
<tr>
<td>1.1.2 SMD PCBA manufacturing</td>
<td>10</td>
</tr>
<tr>
<td>1.1.3 Component supply</td>
<td>12</td>
</tr>
<tr>
<td>1.1.4 High level planning</td>
<td>13</td>
</tr>
<tr>
<td>1.1.5 Shop floor scheduling</td>
<td>13</td>
</tr>
<tr>
<td>1.2 NEW PLANNING PROCESS</td>
<td>15</td>
</tr>
<tr>
<td>1.3 SCOPE</td>
<td>16</td>
</tr>
<tr>
<td>1.4 STAKEHOLDERS</td>
<td>17</td>
</tr>
<tr>
<td>1.5 RESEARCH QUESTION</td>
<td>18</td>
</tr>
<tr>
<td>2 PROOF OF CONCEPT</td>
<td>19</td>
</tr>
<tr>
<td>2.1 COST FUNCTION</td>
<td>19</td>
</tr>
<tr>
<td>2.2 SOFTWARE APPLICATION</td>
<td>20</td>
</tr>
<tr>
<td>2.3 RESULTS</td>
<td>21</td>
</tr>
<tr>
<td>3 DEVELOPMENT OF COST FUNCTION</td>
<td>23</td>
</tr>
<tr>
<td>3.1 PARTIAL COSTS</td>
<td>23</td>
</tr>
<tr>
<td>3.1.1 Changeover cost function</td>
<td>26</td>
</tr>
<tr>
<td>3.1.2 Feeder preparation cost function</td>
<td>27</td>
</tr>
<tr>
<td>3.1.3 Warehouse cost function</td>
<td>28</td>
</tr>
<tr>
<td>3.1.4 Tardiness cost function</td>
<td>29</td>
</tr>
<tr>
<td>3.2 FINAL COST FUNCTION</td>
<td>29</td>
</tr>
<tr>
<td>3.3 EVALUATION OF THE COST FUNCTION</td>
<td>30</td>
</tr>
<tr>
<td>4 EXACT OPTIMIZATION METHODS</td>
<td>31</td>
</tr>
<tr>
<td>4.1 METHODS TO SOLVE THE PROBLEM</td>
<td>31</td>
</tr>
<tr>
<td>4.1.1 Mixed integer linear programming (MILP)</td>
<td>32</td>
</tr>
<tr>
<td>4.1.2 Column generation</td>
<td>32</td>
</tr>
<tr>
<td>4.2 MODELS OF THE PROBLEM</td>
<td>33</td>
</tr>
<tr>
<td>4.2.1 Three-field notation</td>
<td>33</td>
</tr>
<tr>
<td>4.2.2 Mixed integer program (MIP)</td>
<td>33</td>
</tr>
<tr>
<td>4.2.3 Set partitioning formulation</td>
<td>35</td>
</tr>
<tr>
<td>4.3 ANALYSIS OF TIME COMPLEXITY</td>
<td>35</td>
</tr>
<tr>
<td>5 HEURISTIC OPTIMIZATION ALGORITHMS</td>
<td>37</td>
</tr>
<tr>
<td>5.1 SCHEDULE GENERATION METHODS</td>
<td>37</td>
</tr>
<tr>
<td>5.2 NEIGHBORHOOD STRUCTURE</td>
<td>38</td>
</tr>
</tbody>
</table>
5.2.1 Specific neighborhood structures ...................................................... 39
5.2.2 Neighborhood size ........................................................................... 40
5.3 Visualization of cost change ................................................................. 41
5.4 Local search (LS) ................................................................................. 42
5.5 Tabu search (TS) .................................................................................... 43
5.6 Simulated annealing (SA) ................................................................. 45
   5.6.1 Cooling scheme .............................................................................. 46
5.7 Genetic algorithm (GA) ......................................................................... 47
   5.7.1 Initialization .................................................................................. 48
   5.7.2 Selection methods ........................................................................... 48
   5.7.3 Crossover methods ......................................................................... 48
   5.7.4 Mutation methods ......................................................................... 49
5.8 Overview ................................................................................................. 49

6 Implementation and testing environment ..................................................... 51
   6.1 Benchmark environment ................................................................ 51
   6.2 Test and validation sets ................................................................... 51

7 Evaluation of heuristic algorithms ............................................................ 56
   7.1 Local search methods ........................................................................ 56
   7.2 Tabu search ......................................................................................... 61
   7.3 Simulated annealing .......................................................................... 66
   7.4 Genetic algorithm .............................................................................. 69
   7.5 Final results ......................................................................................... 72
   7.6 Hyper-parameter tuning .................................................................... 75

8 Conclusions and recommendations .......................................................... 76
   8.1 Research questions ............................................................................ 76
   8.2 Further conclusions .......................................................................... 77
   8.3 Recommendations ............................................................................. 78
   8.4 Future research .................................................................................. 79

References .................................................................................................. 80

Appendices ................................................................................................. 83
   A Data analytics ......................................................................................... 84
      A.1 Data warehouse ................................................................................. 84
      A.2 Feeder movement .......................................................................... 85
      A.3 Feeder preparation ....................................................................... 87
      A.4 Warehouse .................................................................................... 88
   B Pseudo-code algorithms ........................................................................ 90
   C Example scheduling application .......................................................... 94
   D Theoretical and practical research ...................................................... 96
      D.1 Integer programming methods ....................................................... 96
      D.2 Local search .................................................................................... 97
      D.3 Meta-heuristics ............................................................................. 98
Abstract

In the graduation project presented in this thesis an application driven optimization problem is treated. The practical problem concerns the costs of a particular printed circuit board assembly (PCBA) process at Prodrive Technologies. These costs strongly depend on the schedule that is used for the production process. This schedule was previously developed by shop floor planners without any algorithmic help. The main topic of this thesis is the development of an algorithmic approach for this schedule optimization problem.

We build a bridge between a practical scheduling problem at Prodrive Technologies and several optimization algorithms known from the literature on medium scale multi objective parallel machine scheduling problems.

A proof of concept is developed which implements the tabu search algorithm using a 2-opt neighborhood structure. This algorithm is used to minimize a relatively simple cost function over the bill of material of the products that have to be scheduled. The proof of concept considers a single production line for which all jobs must be split over the production line by hand. The software application that has been developed as a proof of concept was introduced on the production floor and the results are promising.

After the proof of concept a more sophisticated cost model is developed. This cost model takes more aspects that are relevant for the costs of the production process into account. It not only includes the costs of the PCBA production environment, but also other relevant ones, such as component preparation and warehouse picking costs. It turns out that exact evaluation the cost function is computationally very expensive.

For this more advanced cost model, both exact and heuristic optimization methods are investigated. Only the heuristic algorithms are implemented and tested. For the exact methods a MIP model and set partitioning formulation are derived. Furthermore, an analysis of the time complexity is given, which shows that the optimization problem is NP-hard. Based on this analysis we expect that the exact optimization methods are not feasible in practice and therefore we did not implement these.

The following classes of heuristic algorithms are studied: local search, iterated local search, tabu search, simulated annealing and genetic algorithms. For the performed experiments the neighborhood structures job jump, job swap, 2-opt and 3-opt are used. For simulated annealing a linear and exponential cooling scheme is applied. Furthermore we introduce and study heuristic algorithms that are based on an approximate evaluation of the costs, to reduce the computational work per iteration.

Based on a systematic experimental study we give several recommendations. For schedules wherefore only a short optimization time is available, a local search algorithm that uses an approximate evaluation of a job jump neighborhood structure is recommended. For longer runs of six hours the tabu search algorithm and simulated annealing algorithms perform similarly. For the tabu search algorithms an approximate evaluation of a neighborhood structure constructed out of job jumps is recommended. An exact evaluation of a job jump neighbor and an exponential cooling scheme is recommended for simulated annealing.
# List of Abbreviations

<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AOI</td>
<td>Automated Optical Inspection</td>
</tr>
<tr>
<td>AOSI</td>
<td>Automated Optical Solder Joint Inspection</td>
</tr>
<tr>
<td>BOM</td>
<td>Bill of Material</td>
</tr>
<tr>
<td>CAL</td>
<td>Conventional Assembly Line</td>
</tr>
<tr>
<td>CSV</td>
<td>Comma Separated Values</td>
</tr>
<tr>
<td>DPI</td>
<td>Dimensional Paste Inspection</td>
</tr>
<tr>
<td>FIFO</td>
<td>First In First Out</td>
</tr>
<tr>
<td>FTE</td>
<td>Full Time Equivalent</td>
</tr>
<tr>
<td>GA</td>
<td>Genetic Algorithm</td>
</tr>
<tr>
<td>GUI</td>
<td>Graphical User Interface</td>
</tr>
<tr>
<td>IQR</td>
<td>Inter Quartile Range</td>
</tr>
<tr>
<td>LEA</td>
<td>Logistic Execution Application</td>
</tr>
<tr>
<td>LM</td>
<td>Laser Marking</td>
</tr>
<tr>
<td>LS</td>
<td>Local Search</td>
</tr>
<tr>
<td>MIP</td>
<td>Mixed Integer Program</td>
</tr>
<tr>
<td>MILP</td>
<td>Mixed Integer Linear Program</td>
</tr>
<tr>
<td>NP</td>
<td>Nondeterministic Polynomial time</td>
</tr>
<tr>
<td>PCB</td>
<td>Printed Circuit Board</td>
</tr>
<tr>
<td>PCBA</td>
<td>Printed Circuit Board Assembly</td>
</tr>
<tr>
<td>PNP</td>
<td>Pick and Place</td>
</tr>
<tr>
<td>SA</td>
<td>Simulated Annealing</td>
</tr>
<tr>
<td>SE</td>
<td>Software Engineer</td>
</tr>
<tr>
<td>SMD</td>
<td>Surface Mount Device</td>
</tr>
<tr>
<td>SMT</td>
<td>Surface Mount Technology</td>
</tr>
<tr>
<td>SPR</td>
<td>Solder Paste Printing</td>
</tr>
<tr>
<td>TS</td>
<td>Tabu Search</td>
</tr>
<tr>
<td>TSP</td>
<td>Traveling Salesman Problem</td>
</tr>
<tr>
<td>WPE</td>
<td>Work Preparation Engineer</td>
</tr>
</tbody>
</table>
### List of Terminology

**Candidate solution**
A candidate solution is an element of the search space. It does not have to be a good solution but it has to satisfy all constraints.

**Changeover time**
The changeover time is the period required to changeover a production unit from the production of one product type to the production of another product type.

**Chromosome**
A chromosome is a single solution out of the population of a genetic algorithm.

**Bill of material**
The bill of material is the list of components that are needed for the composition of a single product.

**Feeder**
A feeder is a piece of physical hardware that can be placed in a machine to feed it with components.

**Feeder carrier**
A feeder carrier is a movable rack that can be used to move a set of feeders. It tracks the presence of feeders.

**Fitness**
The fitness of a solution, often referred to as fitness function, denotes the outcome of an evaluation of the cost function for the given solution.

**Global minimum**
The global minimum is the global optimum for a minimization problem.

**Global optimum**
The global optimum is a solution for which the evaluation of the cost function is better than all other feasible points.

**Hill climbing**
Hill climbing is an optimization technique. It attempts to improve the selected solution by applying an incremental change to it.

**Incumbent solution**
The incumbent solution denotes the best feasible solution found so far.

**Local minimum**
The local minimum of a problem is the local optimum of a specific type of problems, namely, minimization problem.

**Local optimum**
A local optimum is a solution which is optimal within its neighborhood.

**Makespan**
The makespan is a time interval that is measured from the start of the first starting job till the end of the last finishing job.

**Material container**
A material container is defined as a container that can hold certain material. Examples of material containers are feeders, trays and bags.

**Neighborhood**
A neighborhood of a point is a subset of the search space that exist out of feasible solutions and that are in some sense close to the original point.

**Pipelining**
Pipelining is the division of operations in multiple steps so that they can be executed on different targets in parallel. A general goal of pipelining is to increase the overall throughput of a process or system.

**Production unit**
A production unit is a group of machines that are often used together to create products of a specific class.
<table>
<thead>
<tr>
<th>Term</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>Search space</td>
<td>The search space is the set of feasible points for an optimization problem.</td>
</tr>
<tr>
<td>Setup time</td>
<td>The setup time is the period required to setup a production unit for the production of a specific product type.</td>
</tr>
<tr>
<td>Solution space</td>
<td>The solution space is exactly the same as the search space.</td>
</tr>
<tr>
<td>Storage type</td>
<td>The storage type denotes the type of product that is stored.</td>
</tr>
<tr>
<td>Surface mount technology</td>
<td>Surface mount technology is a technique that is used to produce surface mount devices.</td>
</tr>
<tr>
<td>Surface mount device</td>
<td>A surface mount device is an electronic device existing out of a printed circuit board where on components are mounted.</td>
</tr>
</tbody>
</table>
Chapter 1

Introduction

This document describes a graduation project for the master Embedded Systems at the Technical University of Eindhoven. The project concerns a complex scheduling problem at the company Prodrive Technologies in Eindhoven. A first objective of this project is to prove that switching from a hand-made schedule to an algorithmic solution is feasible in practice. A second objective is to create a realistic model of the underlying production process. A third and main objective is to investigate a class of algorithms that can be applied on the created model.

The investigated algorithms cope with a non-identical parallel machine scheduling problem with a sequence dependent cost function. The objective is to find one or more algorithms that can be applied for scheduling a production process performed in the current and future manufacturing factories of Prodrive Technologies.

Extensive research is available on related theoretical scheduling problems [1, 2]. Furthermore there is a lot of considerable research on related scheduling problems such as job shop, flow shop and open shop scheduling. However, as far as we know no research exists that tackles the particular scheduling problem that arises at Prodrive Technologies in a systematic and substantiated manner.

The production process concerns the placement of components using surface-mount technology (SMT) on printed circuit boards (PCBs). A component that is mounted using this technology is called a surface-mount device (SMD) and a PCB where on one or more SMDs are placed is called a printed circuit board assembly (PCBA) in the manufacturing industry.

In Section 1.1 we provide some background information on the problem and the related production process. We start with a description of the required background knowledge about Prodrive Technologies, the production process and the current scheduling approach. In Section 1.2 a new (high level) planning process is described which will be introduced at Prodrive Technologies in the future. In this project we aim at a systematic approach, therefore at the end of this chapter, in Sections 1.3 to 1.5, the scope is given, stakeholders are described and finally four concrete research questions are given.

1.1 Background

For a better understanding of the practical problem some background information about the company Prodrive Technologies is given. In this section first some basic information about Prodrive Technologies is presented. Then the SMT production process is described. Thereafter the component supply and logistics division that arranges this supply are described. Finally a description of the current planning and manual shop floor scheduling procedures are given.

1.1.1 Prodrive Technologies

Prodrive Technologies, a company established in 1993, is one of the fastest growing privately owned companies in Europe. Back in 1993 only electronic designs were created and some product delivery was performed. The whole manufacturing and supply chain process was outsourced.
Concerning human resource, Prodrive Technologies has grown significantly over the past 25 years. The number of employees at Prodrive Technologies converted into a full-time equivalent (FTE) is approximately 1600. From these employees 450 FTE are working in the Research and Development department. Prodrive Technologies focuses on autonomous growth and a solid preservation of our company culture. The company culture strongly stimulates equality, trust and responsibility values.

Geographically, Prodrive Technologies has tremendously grown as well. Currently multiple manufacturing plants exist in Eindhoven, a manufacturing plant is running in Suzhou (China) and also a manufacturing plant is currently being build in Boston. Furthermore sales offices are placed in more than 5 countries and product service is performed from multiple countries as well.

Currently the core business consists of designing electronics, software, mechanics and manufacturing services. Furthermore lately there is more focus on of-the-shelf products. Products and services for these core businesses and of-the-shelf-products are developed within technology programs. One of these technology programs is Industrial Automation, which aims at creating the factory of the future.

More information about Prodrive Technologies can be found on the website [3].

One of the main steps in the production process of most products developed by Prodrive Technologies is SMT component placement. This step in the production process is performed by the SMD PCBA manufacturing division. This division employs more than 100 FTE and is one of the largest internal customers of the warehouse division. Since the equipment used at the SMD PCBA manufacturing division is very expensive, this division is responsible for the major part of the production costs of Prodrive Technologies. If optimizing this production process turns out to be feasible, it will amplify the overall growth of Prodrive Technologies.

1.1.2 SMD PCBA manufacturing

At the manufacturing factories of Prodrive Technologies, a large number of different types of printed circuit boards is produced. The variety in the product types is caused by two reasons. Firstly, a lot of different prototypes are produced for testing products that are developed in-house by the research and development department. These orders are often called proto-orders. Secondly, small orders of a huge variety of products are sold by the sales division.

All products whereon components must be placed using SMT are processed by a surface mount device (SMD) production line. These SMD production lines are very expensive and each line requires multiple
operators to keep it running. All SMD production lines are located at the SMD PCBA manufacturing division of the operations department.

At Prodrive we describe the production process performed by the SMD production lines as a set of operations that are performed consecutively on the products that are processed by the SMD production line. Currently each SMD production line performs the following operations, in the given order:

- Product marking / laser marking (LM)
- Solder paste printing (SPR)
- Dimensional paste inspection (DPI)
- SMD component placement / pick and place (PNP)
- SMD placement inspection / automated optical inspection (AOI)
- Reflow soldering (RSO)
- SMD solder joint inspection (AOSI)

Some operations of these operations are performed by multiple machines. For example, the PNP operation is performed by 10 consecutive Fuji NXT II machines.

Most of the products produced by the SMD production lines also require through-hole technology. Therefore, after finishing the SMD PCBA manufacturing production process, these products are handed over to the Conventional PCBA manufacturing division. This division uses another line of machines, namely the conventional assembly line (CAL).

Since the products first need to be produced at the SMD PCBA manufacturing division, the Conventional PCBA manufacturing division uses a separate production schedule. The scheduling problem for the Conventional PCBA manufacturing division is expected to be, on an abstract level, similar to that of the SMD PCBA manufacturing division. On a lower level, it performs another set of operations than performed at the SMD production lines and therefore certainly has other decision variables and another cost function structure.

To provide a better understanding of how an SMD production line works, an example of the composition of such a production line is illustrated in Figure 1.1. In this example, the machines in a single SMD production line are shown in the correct order. Furthermore some possible operator actions are indicated. Finally some products and their locations in the SMD production line are shown. Note that this figure presents an extreme simplification of reality.

A schedule is only usable in practice if the deviation between the expected- and actual time of the scheduled activities is not too large. Currently the largest difference between the expected- and actual changeover time of a product side is caused by necessary changes of the feeder arrangement in the PNP machines. A feeder is a portable piece of hardware that can hold a tape, which can contain components. These components are placed on the PCBs by Fuji PNP machines. These feeders are placed manually in the PNP machines by operators. It is expected to keep being a manual task in the coming decades.

Before a PNP machine starts its operations on a PCB the required feeders and trays must be present. Which components are placed by each PNP machine for a product type is determined by a program that is created prior to production. These programs can be created with a specific machine configuration. In this machine configuration a configuration type can be given for which the options A, B and AB exist.

When configurations A is used, a program is created for which all feeders must be placed on the left half of the feeder racks in the PNP machines. Similarly when using configuration B, a program is created for which all feeders must be placed in the right half of the feeder racks in the PNP machines. When using these configurations and producing them in an alternating order, changeover can be performed in parallel with production but the PNP machines run a little slower. When the configuration type AB is used, the feeder requirement is spread over the whole feeder rack.

At Prodrive regularly high priority orders disturb the planned schedule. Therefore it is important that it is possible to schedule a high priority order. Furthermore rescheduling of orders for which
operators of the SMD PCBA manufacturing division are already performing preparation is considered disturbing for the operators and must therefore be avoided. Also rescheduling orders for which the feeder preparation subdivision has already performed preparation is considered disturbing.

From experience gained at Prodrive it is known that scheduling problems at the SMD PCBA manufacturing division result in more scheduling problems for the conventional PCBA manufacturing division. This is because when the variation between the expected and actual production time at SMD increases, the expected release date for the Conventional PCBA manufacturing division becomes less reliable.

All components that must be placed on a single product to compose a final product are in a list called the bill of material (BOM). In this report we consider the BOM as the list of components that are placed by the SMD PCBA manufacturing division. It is filtered beforehand and all components are listed only once, since only a single feeder is required for production.

1.1.3 Component supply

Before products can be produced at the SMD PCBA manufacturing division, components have to be placed in the SMD machines. The supply of these components introduces workload for multiple divisions of the operations department. This work can increase or decrease when different schedules are used. This section describes how the tapes and trays received at the goods receipt end up in the machine and move through the factories of Prodrive Technologies.

All components that are used by the SMD PCBA manufacturing division are provided in material containers. Currently we use three types of material containers namely tapes, trays and bags. The purchasing division aims at ordering as many components as possible on tapes since trays are harder to trace and the components in bags need to be placed in tapes before they can be used by the SMD machines. The most expensive components are normally delivered in trays. These trays always have the same size.

The significant part of the components is provided on tapes. Tapes of multiple sizes exist, more specifically, we use storage type 011 (L), storage type 012 (XL) and storage type 013 (XXL).

The component supply to the SMD production lines is handled by two divisions namely Logistics and SMD PCBA manufacturing. At the logistics division, the relevant subdivisions are the goods receipt, central warehouse and SMD PCBA warehouse. At the SMD PCBA manufacturing division, only the subdivision feeder preparation is part of the component supply chain. These subdivisions all need to
run properly so that the correct components will reach the SMD production lines in time. Figure 1.2 shows schematically the component flow through the described subdivision.

At the subdivision goods receipt, everything that arrives at Prodrive Technologies is received. This does not only include components for SMD PCBA manufacturing division but also the components for all other divisions of the operations department, products that must be serviced and supply for other departments in the organization.

All components that are sourced for the operations department and products that must be serviced, are always first stored in one of the central warehouses. The retrieval and storage of tapes is traced in SAP R/3 Enterprise. Using this system, at each storage location, for each component type the number available pieces can be tracked and traced. In SAP a transfer order item refers to the movement of one or multiple components of the same type to another storage location.

When the component availability is too low in a particular SMD PCBA Warehouse, a new batch of these components is moved from one of the central warehouses to this one. This is arranged by the operators of the logistics division. These movements are also traceable in SAP.

When an order is scheduled to be produced at SMD PCBA manufacturing, the logistics division picks the required components from the SMD PCBA warehouse. This includes all components that are not in one of the SMD PCBA manufacturing subdivisions, yet. They are placed in boxes and delivered at the feeder preparation subdivision. Furthermore components that are no longer needed at production are retrieved from the feeder preparation subdivision. These components are placed back in the SMD PCBA warehouse for later use.

At the feeder preparation subdivision, tapes are placed in and removed from feeders. Feeders that are prepared are placed in feeder carriers. Feeder carriers that contain all required feeders needed for an order that is not on the line yet, are placed in the waiting area for the concerning SMD production line.

When the component placement for a production order is completed, the feeders that are no longer needed at the SMD production line are placed in a feeder carrier, which is returned to the feeder preparation subdivision. If feeders containing tapes are returned from an SMD production line and are needed in one of the next orders on another SMD production line, they are placed in feeder carriers in the staging area. The feeders holding tapes that are not needed for scheduled orders are disassembled, meaning that tapes are removed from their feeders. The removed tapes are placed in boxes for return to the SMD PCBA warehouse and the feeders become available for the binding of new tapes.

### 1.1.4 High level planning

At Prodrive Technologies a high level planning is created for all departments. This planning defines which orders are produced when, for example in which week, and at which department. During the development of this planning, the realization of end products is split into different orders that are spread over multiple departments. The planning is created based on the chase demand strategy.

For the SMD PCBA manufacturing division buckets of orders are created that are to be scheduled within a specific time window. Currently these buckets exist of approximately 100 orders and the time window is on average 1 week.

During the rest of this report these orders are often referred to as jobs, since this is a more abstract term, that is commonly used in theoretical scheduling papers and books.

When a bucket of jobs is ready for production by the SMD PCBA manufacturing division, it is handed to the shop floor planner. This planner collaborates directly with the production team and schedules jobs on the time-lines for the SMD production lines. Note that the buckets of jobs may come with individual deadlines and release times per job.

### 1.1.5 Shop floor scheduling

Currently the schedule developed by the shop floor planner is manually entered in an in-house developed software application, the so-called Planning Viewer. This is done by shop floor planners that know a lot about the products that are produced and about the SMD production process. In the
Planning Viewer the top and bottom side of a PCB are planned separately. Furthermore, before each scheduled product side, setup time can be added. This additional time is needed for the operators to configure the SMD production line for the new product side.

Note that there is a difference between setup time and changeover time in a manufacturing schedule. Setup time indicates the required time for the preparation for the production of a product type. Changeover time is the time needed to change between the production of two product types. Since the operators at the SMD production lines work on tearing down the lines from the last produced product type at the same time as preparing for the next product type, setup time is not sufficient enough. Therefore changeover times should be used in the schedule instead.

Also note that the Planning Viewer does not take individual operations into account. It schedules the expected production time for a single side of the product, for all operations in the SMD production line. This production time is a manual estimation and therefore, it is sometimes far from accurate.

The schedule created in the Planning Viewer is an extreme simplification of the reality and is strongly dependent on the shop floor planner's experience. Furthermore, the number of products and the number of production lines are increasing, which makes it very difficult for a single person to take the whole system into account.

In recent measurements, it can be seen that the average utilization/up-time of the production lines fluctuates between 25% and 35%. Since the investment in these SMD production lines are high, a higher utilization is highly profitable. The low utilization is partially caused by the large variety of distinct products that are produced. Another inhibiting factor for a high utilization is the average order quantity, which is lower than that of competitors, causing relatively more setup time. Nevertheless the manufacturers of the Fuji PNP machines that are used in the SMD production line indicate that an up-time of 60% should be feasible for our production needs. Therefore, it is expected that creating a better schedule will result in a higher utilization, which motivates this project.

Approximately two years ago a project called ‘Manufacturing Planning’ was started at Prodrive Technologies. The goal of this project was to develop a new software application that should provide the ability to develop manual and automated schedules for all departments. Approximately 6000 software engineering hours were spent on this project and basic schedule viewing functionality has been implemented. Unfortunately at that time nobody knew how to develop automated scheduling algorithms for our operations department. Therefore this project was terminated and has not been continued since.

Because of the described low utilization and the already failed attempt to solve the issue, this graduation project is of high relevance for Prodrive. The previous attempt has shown that this issue is very complex. Now the priority is set on finding a feasible optimization algorithm. The focus is on this algorithmic development and we expect that the implementation of a scheduling application is less difficult.

As previously mentioned, the Planning Viewer is very general and lacks information, which is crucial for a good schedule. The following problems are identified:

- **Parallel material container requirement**: when there are multiple products that need the same component, in certain cases, they cannot be produced at the same time. This is because there is only a single material container available. Currently there is no recipe to deal with this problem.

- **Production line constraint**: some products can only be produced on a subset of the SMD production lines; for example, because they have a wider panel that does not fit through every line, or they need nitrogen in the re-flow soldering oven, which is only present at certain lines. That an order is scheduled on a specific line because of this constraint is not enforced by the Planning Viewer.

- **Pipelining**: when the last product of an order has passed the first machine but is still in the line, the machines before this last product, can already be setup for the next order. This means that the setup time is related to the line latency of the previous order. Therefore the setup time should actually be replaced by a changeover time. The Planning Viewer does not consider this when scheduling setup time.
● **Available workforce**: operators are scheduled to come to work for a specific amount of time on each day. This means that there is a specific operator capacity at each specific moment in time. At the moment, the amount of parallel work that is required to keep the production lines running over time is unknown. This means that when there are peaks in the amount of work that needs to be performed, there will not be enough operators present (since this arises unannounced). This may cause one or more lines to stop. This can be avoided by trying to approach an approximately constant distribution of the amount of work over time. The Planning Viewer has no knowledge of the amount of operators or the amount of required work.

● **PNP machine configuration**: each PNP machine has three configurations in which feeders can be placed (A, B and AB). This configuration has a significant impact on the setup and production time of an order. This is because when using A and B in consecutive orders, the machine can be configured for the second order while the production is still running. Currently solely AB is used.

● **PNP group setup**: a setup for multiple products on a single line can be made in the fujitrax software. This group setup will minimize the setup time because it will allow the operators to configure the PNP machines for the whole group. This is limited to the slot size per PNP machine (a group setup for which the number of component types is higher than the number of slots in the PNP machines is not feasible). The Planning Viewer does not indicate group setups.

### 1.2 New planning process

The Planning division of the operations department is developing a strategy for changing the whole planning cycle. These changes are expected to be introduced in Q1 2021. Creation of this high-level planning cycle is not in the scope of this graduation project. However this project should consider the change in the higher level planning process and its implications. Therefore this new high level planning process and its goals are briefly explained.

The goal of the new high-level planning cycle is to obtain more scheduling freedom per department so that better schedules can be developed. More scheduling freedom can be reached, for example by earlier knowing what to schedule, so that schedules are more flexible. One further goal is to decrease the amount of incidents and escalations by providing a more realistic planning. Figure 1.3 shows the objective, constraints and outcome of this new high-level planning cycle.

<table>
<thead>
<tr>
<th>Now</th>
<th>Scheduling Gantt</th>
<th>Bucketized planning weekly buckets</th>
<th>Bucketized planning monthly buckets</th>
</tr>
</thead>
<tbody>
<tr>
<td>Objective:</td>
<td>Making feasible production plan satisfying production order due dates while minimizing changeover, week picking workload and local stock.</td>
<td>Making feasible weekly production plan satisfying all demand and safety stocks and minimizing stock and stock movements.</td>
<td>Making feasible monthly production plan satisfying all demand and safety stocks and minimizing stock and stock movements.</td>
</tr>
<tr>
<td>Constraints:</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Work center availability</td>
<td></td>
<td>Current level = maximum by opening shifts</td>
<td>Current level = maximum level after training or hiring</td>
</tr>
<tr>
<td>Human resource availability</td>
<td></td>
<td></td>
<td>Resource availability</td>
</tr>
<tr>
<td>Earliest start date production order</td>
<td></td>
<td></td>
<td>Component availability given load time components</td>
</tr>
<tr>
<td>Product and changeover times</td>
<td></td>
<td></td>
<td>Central and local workstation capacity</td>
</tr>
<tr>
<td>Safety time</td>
<td></td>
<td></td>
<td>Avg production and changeover times</td>
</tr>
<tr>
<td>Fixed released production orders at start of the horizon</td>
<td></td>
<td></td>
<td>Minimum lot sizes and rounding values</td>
</tr>
<tr>
<td>Outcome:</td>
<td></td>
<td></td>
<td>Standard waiting time</td>
</tr>
<tr>
<td>Production schedule</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Figure 1.3: A figure of the new high-level planning process proposed by the planning division (internal report).

In this figure we observe that the buckets that must be produced for the next four week are known by the concerning production division. Since this concerns a longer period than currently available, when
scheduling in the future, it is possible to schedule the last planned jobs and the new bucket of jobs in a better way. Actually at the moment a new bucket of jobs is ready to be scheduled, the jobs that have to be produced in the next four and a half weeks are known. Since the next two days should not be changed on the schedule of the SMD PCBA manufacturing division, four weeks of jobs can be re-optimized every week. Note that the next two days should not be changed since the logistics division may have already performed some preparation.

Note that rescheduling all jobs of the next four weeks will not always be beneficial in the shop floor scheduling process. This is because rescheduling everything may change too much in the schedule. This also creates obvious disadvantages. Therefore these new schedules are only expected to be applied if the cost function is significantly decreased.

1.3 Scope

Since numerous algorithms exist for traversal through the solution space to scheduling problems and countless options can be included and excluded from the to be designed cost function, it is crucial that this graduation project is scoped in a clear manner. This section describes the scope of this project by placing certain specific technical possibilities outside of the scope.

- **SMD PCBA manufacturing division**: it is expected that the abstract approach of this project can later be reused for other divisions, such as the conventional PCBA manufacturing division. Investigating the required cost functions for both divisions at this moment is considered to be too much work and therefore risks achieving a good optimization algorithm. Furthermore numerous other division exist for which schedules can be optimized. Currently the SMD production lines are the most expensive equipment in Prodrive. Furthermore the profit of the SMD PCBA manufacturing division is the highest. Therefore, the schedule of the SMD PCBA manufacturing division has been selected for this project and all other divisions are considered out of scope.

- **PNP group setup**: when the prior described group setup is tackled in the same optimization problem and cost function as the scheduling problem, a too large optimization problem arises. Therefore they are considered as two separate optimization problems. Since creating a good group setup proposal for the schedule can be seen as a whole separate graduation project and optimizing a good schedule is currently considered more important, group setup proposals are considered out of scope for this project.

- **Product side separation**: as described in the previous sections production can be performed on top and bottom sides of products. Also the Planning Viewer allows separate scheduling of these sides. Optimizing these product sides separately introduces a precedence relation in the model. This relation complicates the scheduling model and can be avoided by always producing them consecutively on the same SMD production line. Since scheduling the top and bottom side apart is expected to introduce more work for the operator of the SMD PCBA manufacturing division and increases the changeover time in nearly all cases, we decided to always schedule them together. Therefore all products that need work on two sides are considered as one job in which top processing it followed by bottom processing. Since the model is created like this, algorithms that investigate scheduling problems with precedence relation are considered out of scope.

- **PNP machine configuration**: the Fuji PNP machines can work in a configuration that can be set in three manners, namely A, B and AB mode. Introducing these different configurations will significantly complicate the model since this may introduce multiple manners of scheduling a single job and therefore a new dimension will be introduced to the change over time of the model. Since introducing use of this functionality on production will require a new process for the operators on the SMD production lines, this hugely complicates the introduction in reality. Therefore it was decided not to include this in the model and place it out of scope.

- **Parallel material container requirement**: taking the described parallel use of components by jobs into account when too few tapes are available, creates an inter machine constraint that is hard to evaluate when making small changes to a solution. Furthermore a risk arises that can cause problems when the schedule for a single line is delayed by problems on the production
If this happens tapes may be needed by multiple machines in parallel again. Rather than complicating the scheduling problem in this manner, we decided to tackle this issue in a different way. In the conclusion we formulate a recommendation to create insight in the requirements of parallel tapes. Using this, the logistics, SMD PCBA manufacturing and purchasing division can work together and make sure that enough tapes are present in the SMD PCBA warehouse. This can be done by retrieving multiple tapes from warehouse, buying additional tapes or splitting existing tapes.

- **Precondition:** With a precondition we mean that there are already orders on the schedule, wherefrom changeovers must be performed to the first newly added job on each SMD production line. Considering the preconditions (based on the existing schedule) tends to be hard when working with exact scheduling models. Furthermore, it introduces additional complexity, which does not per definition lead to a better solution in practice. This is especially the case for scheduling problems for which finding a global optimum is not tractable. Therefore, for some of the exact optimization models treated in Chapter 4 most preconditions are ignored and removed from the problem.

- **Software implementation:** To be able to extensively study the available algorithms during this project, the implementation of a software application is outside of the scope for this project. However, introduction to the SMD PCBA manufacturing division is kept in mind. Since previous projects have proven that changing too much in the way of working often results in an infeasible switch for the SMD PCBA manufacturing division.

### 1.4 Stakeholders

This section describes the stakeholders for this graduation project. Furthermore some of their concerns are discussed. The identified stakeholders are management, planning, shop floor planning, SMD PCBA manufacturing, feeder preparation, logistics and software.

- **Management:** A main concern of the management team is to increase the total throughput of the SMD production lines so that the revenue of Prodrive Technologies can be increased by accepting more orders. A second concern is to decrease the cost in the form of work effort of the SMD PCBA manufacturing division, to save costs in the operations department.

- **Planning:** The main concern of the planning division is to have an easy and flexible way to create schedules for testing if a bucket of orders is feasible. Furthermore their concern is to increase the feasible load on the SMD PCBA manufacturing division so that they have more freedom in creating weekly buckets of orders for this division.

- **Shop floor planning:** Shop floor planning has the concern that they must be able to move single orders around to work around incidents. Furthermore the possibility to scheduling multiple orders together in one action is a prerequisite. Also they want to have all the features in the current scheduling tool in place (or another tool providing at least similar functionality).

- **SMD PCBA manufacturing:** The SMD PCBA manufacturing division has the concern that no orders are scheduled on the wrong SMD production line. Furthermore they will only accept changes when the work process does not change too much and that a graphical visualization of the current schedule is available. This visualization must also include the actual production progress as well as the created schedule.

- **Feeder preparation:** An important concern of the feeder preparation subdivision is that the number of tapes that have to be prepared does not increase too drastically, because increasing the workload asks for an extension of the size of this subdivision. This extension will no longer fit on one floor together with the rest of the SMD PCBA manufacturing division.

- **Logistics:** Similar to the feeder preparation subdivision, the logistics division has the concern that their workload does not increase.

- **Software:** The main concern of the software division of the research and development department is that the described and picked algorithms are implemented in a tractable manner.
other words, preferably no algorithm is picked that requires multiple parallel runners to come to a solution.

1.5 Research question

In this section the main research questions for this graduation project are formulated.

Research question 1:
Can a model-based scheduling process lead to significant improvement in the efficiency of the manufacturing process of the SMD PCBA manufacturing division?

Research question 2:
Which constraints and cost function are useful for modeling our practical problem?

Research question 3:
Is it feasible to apply exact optimization methods to our practical problem?

Research question 4:
Which heuristic algorithms can be applied for the approximate solution of our optimization problem, and how do these perform?
Chapter 2

Proof of concept

Since the planning for the shop floor schedule at the SMD PCBA manufacturing division was previously maintained by hand, we decided to implement a simple proof of concept. The first objective of this prove of concept is to show to the operations department that an algorithmic solution can be beneficial in practice. A second objective is to get used to the environment and dynamics of optimization algorithms.

We decided to create a proof of concept that heuristically optimizes the schedule for a single machine. The first and main reason not to consider multiple machines is that this keeps the model and application simple. Secondly, it is expected that optimizing the current schedule for a single SMD production line can already increase the efficiency of the SMD PCBA manufacturing division. Furthermore a single line solution can be reused for each individual line. Finally, the SMD production line constraints do not have to be considered, since these constraints are evaluated before the proof of concept is applied (these constraints are relevant when the jobs are distributed over the different SMD production lines).

We first describe the cost function that is used in the proof of concept. Secondly we describe the software application that is developed and how this optimizes the cost function. In the last section we give the results on how this software application performs on the production floor.

2.1 Cost function

This section describes the cost function that we use. First a motivation is given why this cost function is expected to increase the overall efficiency of the SMD production lines. Hereafter a mathematical formula of the cost function is presented.

When looking at the SMD production process on the production floor of the SMD PCBA manufacturing division, the (re)placement of material containers causes most downtime of the production lines. This (re)placement partially occurs during the manufacturing of products, but mainly occurs when switching between the production orders.

During a switch between two production orders, material containers with the required components for the next order must be placed. Furthermore material containers containing components that are no longer needed can be removed from the PNP machines. An elementary cost function for the proof of concept is based on minimizing the number of material containers that must be placed and removed. There is no difference made between the type of material containers. Furthermore we use a reasonable simplification, namely that always exactly one material container is needed per component in the BOM of the product types that are produced.

Let \( B_p \) be a set containing the BOM of product \( p \). Let \( J \) be the set of jobs that has to be scheduled and \( p_i \) be the product that has to be produced during job \( i \in J \). Using these definitions the cost function \( c = c(J) \) that we aim to minimize is given by:

\[
  c = \sum_{i=1}^{\lfloor J \rfloor} |B_{p_i} \triangle B_{p_{i+1}}| = \sum_{i=1}^{\lfloor J \rfloor} (|B_{p_i} \cup B_{p_{i+1}}| - |B_{p_i} \cap B_{p_{i+1}}|)
\]
2.2 Software application

This section describes the software application that is made as a proof of concept. We motivate some design decisions and describe the structure of the method which the software application implements.

- We do not implement our method in the Planning Viewer, the application that is currently used to maintain the schedule. This is because this integration would take a lot of time and the shop floor planner can also enter proposed schedules in this application by hand.
- We implement a C# application because this is very convenient for the retrieval of data from the systems currently in place within our infrastructure.
- To spare on software development time we decided to create a console application. This saves the time needed for creating a graphical user interface (GUI). This console application shows the progress and executing activities during a run of the application.
- Rather than entering the jobs that have to be scheduled into the console we read the required input jobs from a CSV file.
- The model can be optimized using exact methods and heuristic methods. It is unclear whether the exact methods will be tractable for the proof of concept. If we consider a sophisticated cost model for the optimization of a multi-line schedule, exact methods will definitely not be applicable. Therefore we optimize the cost function using heuristic algorithms.
- The input jobs are already sequenced and can be used as start for heuristic optimization algorithms. As an alternative we develop a greedy algorithm. This greedy algorithm builds a new sequence by starting with a single job at the start of the schedule and repeatedly selecting a consecutive job. During each iteration the job with the lowest cost increase is added.
- Many heuristic algorithms are available. We pick and implement one algorithm based on previous experience. We do not compare this algorithm to other ones. The algorithm we use is a meta-heuristic algorithm called tabu search [4].
- The tabu search algorithm benefits from a good starting point. Therefore we use the best out of the two starting schedules described above.
- The tabu search algorithm needs to evaluate a set of schedules that are similar to the current schedule. For this a neighborhood structure must be defined. We use 2-opt as neighborhood structure [5].
- The hyper-parameters for this meta-heuristic algorithm are the tabu list length, the neighborhood size and the number of optimization iterations. These hyper-parameters can be put in a configuration file so that they can be changed without the requirement of recompilation. Optimizing these hyper-parameters is not within the scope of this proof of concept, but the responsibility of the process owner of the SMD PCBA manufacturing process.

Figure 2.1 gives a flowchart of the method that is implemented. For performance reasons, first the BOMs of all jobs in the schedule are retrieved. After that a two dimensional matrix is calculated that holds the changeover cost between all possible two jobs. After this a greedy schedule is constructed followed by the start of the tabu algorithm.
### 2.3 Results

The proof of concept has been implemented, tested and handed over to the shop floor planners at the SMD PCBA manufacturing division. This section describes the results obtained by these shop floor planners.

The shop floor planners performed experiments with the new software application, optimizing the order of historical SMD production schedules. Based on the results of these experiments they decided to use this application in their production environment, too. The number of BOM components that have to be (re)placed in the computed schedule are measured. This number is shown in Table 2.1 for three different schedules. These schedules are, the initial schedule, the computed greedy schedule and the best schedule found by the tabu algorithm.

We give a short explanation of the performed experiments:

1. **Three days**: a test case where the schedule of production line one for three consecutive days is placed in the CSV file that is read by the application.

2. **A lot of expected overlap**: a test case where the schedule of line three for one day is placed in the CSV file that is read by the application. The expectation of the operators of the production lines is that there is a lot of overlap in the BOM of the products produced during these orders.

3. **No expected overlap**: the schedule of one day on line five is placed in the CSV file that is read by the application. A shop floor planner and work preparation engineer both expect no overlap in the BOM of the products produced during these orders.

4. **Combined lines**: the present orders on production line one, three and five are placed after each other to create a larger test set.

![Flowchart of the method](image_url)
Table 2.1: Results of experiments performed by the shop floor planners using the prototype application.

<table>
<thead>
<tr>
<th>Description</th>
<th>Production line(s)</th>
<th>Input [#]</th>
<th>Greedy [#]</th>
<th>After tabu search [#]</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 Three days</td>
<td>1</td>
<td>1402</td>
<td>1195</td>
<td>1115</td>
</tr>
<tr>
<td>2 A lot of expected overlap</td>
<td>3</td>
<td>540</td>
<td>434</td>
<td>430</td>
</tr>
<tr>
<td>3 No expected overlap</td>
<td>5</td>
<td>828</td>
<td>792</td>
<td>788</td>
</tr>
<tr>
<td>4 Combined lines</td>
<td>1,3 and 5</td>
<td>3093</td>
<td>2825</td>
<td>2629</td>
</tr>
</tbody>
</table>

These experiments show that for small sequences of jobs, using a greedy algorithm already shows a significant improvement in the number of feeders that have to be replaced. Furthermore running the tabu search algorithm does not show much improvement in these smaller test cases. This may be due to a sub-optimal choice of the hyper-parameters, but most probably it can be explained by the small problem size. When looking at experiment four it can be seen that if the number of jobs increases the tabu search has significantly more effect.

One further (obvious) observation is that optimizing the schedule can decrease the number of actions on the SMD PCBA production lines. This may also lead to a decrease in the workload of the supporting divisions in the operations department. This is due to the fact that less material containers have to be retrieved from and returned to the warehouses.

The successful proof of concept demonstrated the potential of switching from a hand-made schedule to an algorithmic solution. The first step towards an improved algorithmic solution is a more realistic model, as discussed in the next chapter.
Chapter 3

Development of cost function

This chapter describes how the cost function is derived from the underlying practical problem. This is mainly done by manual measurements on the production floor and analysis of the available data sources.

In Section 3.1 we describe how a rough estimation of the costs is made by enumerating and estimating the partial costs. In the same section, we explain how each part is analyzed individually. In Section 3.2 an aggregation of the total cost function is given. Finally in Section 3.3 the time complexity of the cost function is discussed.

3.1 Partial costs

In this section, the total production cost is calculated based on an estimation of the hourly costs. These consist of material depreciation and employment wage costs. An estimation of the current cost is made by the warehouse and SMD team leaders and reviewed with the operators of the production floor. Each cost is briefly described:

1. **Warehouse employee**: at the SMD PCBA warehouses, employees are needed to pick the tapes and trays that contain the components required for production. Furthermore tapes and trays that are no longer needed at production should be transferred back to stock.

2. **Line operator (during changeover)**: during changeover at the production lines, operators change the feeders that are placed in the pick and place machines. Also some tapes and trays are missing and need to be found and gathered. These operators want to get the line running as soon as possible after the previous order was finished.

3. **Line operator (during production)**: during production some tapes may run out of components, components may get stuck in the machine or other problems may occur. The responsibility of these operators is to rebind empty feeders and resolve the problems that occur during production.

4. **Line operator (support)**: besides the pick and place machine, also other machines are needed to keep the SMD process running. The responsibility of the support operator is among others to place new PCB's at the start of the line, change stencils in the SPR machine and resolve non-conformances.

5. **Feeder preparation operator (binding)**: before tapes can be placed in the pick and place machines, they must be placed in feeders. This is done by operators at the feeder preparation subdivision.

6. **Feeder preparation operator (tear down)**: the tapes that are used for production and are no longer must be returned to warehouse. Similar to binding in tapes, this happens at the feeder preparation subdivision.

7. **Team leader (SMD PCBA manufacturing division)**: the SMD team leader determines which users operate on which line, triggers escalations and decides on improvements made on the
SMD machines. Furthermore the status of the preparation of upcoming orders must be overseen.

8. **Team leader (feeder preparation subdivision):** the team leader at feeder preparation is responsible for the on-time delivery of all prepared feeders and trays to the production lines. This role includes directing the binding and tear down operators at feeder preparation and communicating shortages from warehouse.

9. **Work preparation engineer (WPE):** each machine in the SMD production lines requires a custom program per product that is produced. The work preparation is responsible for creating programs for products that have a new release or have never been produced before. Also WPEs provide support to the line operators when problems occur or when maintenance has to be performed.

10. **Software engineer (SE):** software engineers at the SMD PCBA manufacturing division create custom quick win tools that help the operators at the line to improve their productivity.

11. **Line depreciation (during downtime):** each SMD production line is a composition of multiple expensive machines. These machines decrease in value because newer techniques arise every year. Furthermore they wear out after running for a certain amount of hours. These costs do not include the depreciation during production.

12. **Line depreciation (during production):** similar to the depreciation during changeovers, the machine also decreases in value during production.

Note that component consumption is not included in this section, since these are considered to be costs of the product, not of the production process. Furthermore, certain costs are split, based on production and changeover time. This is done because the costs during changeover time are influenced by changing the schedule while the costs during production time are (nearly) independent from the schedule. An overview of the height of these costs can be found in Table 3.1 and Figure 3.1.

<table>
<thead>
<tr>
<th>Description</th>
<th>FTE</th>
<th>Per. mult.</th>
<th>Line mult.</th>
<th>Hourly cost [€]</th>
<th>Total [€]</th>
<th>[%]</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 Warehouse employee</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>4.9%</td>
</tr>
<tr>
<td>2 Line operator (during changeover)</td>
<td>0.37</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>4.6%</td>
</tr>
<tr>
<td>3 Line operator (during production)</td>
<td>0.63</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1.6%</td>
</tr>
<tr>
<td>4 Line operator (support)</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>4.9%</td>
</tr>
<tr>
<td>5 Feeder preparation operator (binding)</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>3.2%</td>
</tr>
<tr>
<td>6 Feeder preparation operator (tear down)</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1.8%</td>
</tr>
<tr>
<td>7 Team leader (SMD PCBA manufacturing div.)</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>0.7%</td>
</tr>
<tr>
<td>8 Team leader (feeder preparation subdivision)</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>0.7%</td>
</tr>
<tr>
<td>9 Work preparation engineer</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>2.1%</td>
</tr>
<tr>
<td>10 Software engineer</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1.4%</td>
</tr>
<tr>
<td>11 Line depreciation (during downtime)</td>
<td>0.29</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>21.5%</td>
</tr>
<tr>
<td>12 Line depreciation (during production)</td>
<td>0.71</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>52.6%</td>
</tr>
</tbody>
</table>

Table 3.1: An estimation of the current hourly costs of the SMD PCBA manufacturing division.
These hourly costs consist of both costs that can be influenced by changing the schedule and costs that are not influenced by changes of the schedule. The costs that cannot be influenced by changing the schedule are 3, 4, 7, 8, 9, 10 and 12. Note that some of these costs may still scale down when the total utilization of the production floor decreases. The costs that can be (partially) influenced by changing the schedule are 1, 2, 5, 6 and 11. Figure 3.2 shows a pie chart of these costs separately.

For quantifying the partial costs of the total cost function we apply a data based analysis. Below we explain this analysis. The line down costs (2 and 11) are described in Section 3.1.1, the warehouse costs (1) in Section 3.1.3, and the feeder preparation costs (5 and 6) in Section 3.1.2.
3.1.1 Changeover cost function

When measuring time on the production floor, the most significant downtime on the line is caused by the Fuji PNP changeovers. In this section we describe how the changeover time and changeover cost per changeover can be calculated. Finally we describe how to calculate the changeover cost per line.

A single PNP changeover on the SMD production line happens in three stages. First, the feeders that are already in the machine are moved to the correct machine. Secondly, newly needed feeders are moved from feeder carriers to the PNP machines, and finally, no longer needed feeders are moved from the machines and placed in feeder carriers.

In this project we relate the number of feeders that have to be moved to the number of distinct components in the BOM of the products between which changeovers are performed. Let \( B_p \) denote the set of distinct components in the BOM of product \( p \), let \( a \) denote the product where from a changeover is performed and let \( b \) denote the product where to a changeover is performed.

We define \( ll_{ab} \) to be the number of feeders that need to change machine on a line for a changeover from product \( a \) to product \( b \). Based on measurements performed on the production line, approximately half of the components that are already on the line are in the correct machine. Furthermore for each element of the BOM a single feeder is in the SMD production line. Let \( cl_{ab} \) be the number of feeders that must be transferred from feeder carriers to the PNP machines for a changeover from product \( a \) to product \( b \) and let \( lc_{ab} \) be the number of feeders that must be transferred from the PNP machines to feeder carriers for a changeover from product \( a \) to product \( b \). Using these definitions an approximation of the number of actions \( ll_{ab}, cl_{ab} \) and \( lc_{ab} \) can be calculated as follows:

\[
\begin{align*}
ll_{ab} &= 0.5 \cdot |B_a \cap B_b| \\
cl_{ab} &= |B_a \setminus B_b| \\
lc_{ab} &= |B_b \setminus B_a|
\end{align*}
\]

The number of actions that are needed to be performed during a changeover between two products is known, but we still have to determine how long such a changeover takes and how much it costs. The time a changeover takes is needed to construct a schedule and to calculate the tardiness cost function, which is described later in this chapter. We calculate this time based on an average time per action. For the cost per changeover we also use the hourly cost that can be found in Table 3.1. The cost of a changeover between two products consists of two parts. During the first part the line is down and therefore this part includes the SMD production line depreciation. During the second part the SMD production line is already running again and therefore only operators must be paid. On the SMD production schedule this part is not incorporated as changeover time. It is incorporated as production time since this part is executed during the production of the next order.

Fortunately the current locations of each feeder in PNP machines can be retrieved from an Oracle database from the Fuji software. Also the location of feeders in feeder carriers can be retrieved from the MSSQL database of an in-house developed application. The database of Fuji is polled with an interval of 500 milliseconds and therefore we have time measurements that can deviate 500 milliseconds from reality. With this data, feeder intervals can be calculated. During this graduation project, a Power BI report is made available which gives insights in these intervals. More information about this Power BI report, how outliers are filtered and scatter plots from which the used averages are retrieved, can be found in Appendix A.2.

When filtering the outliers in the Power BI reports, it can be found that moving one feeder between two machines in the same SMD production line takes 6.51 seconds on average. Similarly, moving a feeder from a feeder carrier to a machine on the line takes 27.56 seconds on average and moving a feeder from a machine on the line takes 165.64 seconds on average. Note that the high time for moving a feeder from a machine on the SMD production line to a feeder carrier can be explained by the manner of working on the production floor. The way of working allows this time to be longer since the SMD production line is running again when these feeders are taken out.

The denoted averages are the differences between the time at which a feeder is taken from its slot and the time at which it is placed in the next slot. Obviously between two of those actions the concerning
operator first needs to look for the next action to perform. Unfortunately no traceability exists to measure the time between two of such actions. This is because it is unknown which operator takes which feeder and mostly more than one operator is performing actions on the PNP machines of a single SMD production line in parallel. Therefore a few manual samples have been measured which result in an average of 11.46 seconds between two consecutive \( l \) actions, 13.71 seconds between two consecutive \( c \) actions, and 13.84 seconds between two consecutive \( lc \) actions.

Note that changeovers are expected to be performed by two people, so we can divide the amount of required time by two.

First we define the changeover time between two jobs. This is the time that the SMD production line is not running between two jobs. Note that the costs of such a changeover is not linear with the time of this changeover because feeders are removed when the line is already running again. Therefore a separate formula is composed for the changeover costs. This formula can be found in Equation (3.3).

We created a separate formula for the changeover time because it is not only a subfunction of the changeover costs but also of the tardiness costs. Therefore we reuse the following formula in Section 3.1.4. Let \( d_{ab} \) denote the changeover time in hours for a changeover from product \( a \) to product \( b \):

\[
d_{ab} = (l_{ab} \cdot \frac{6.51 + 11.46}{60^2 \cdot 2} + c_{ab} \cdot \frac{27.56 + 13.71}{60^2 \cdot 2})
\] (3.1)

Let \( c_{ab} \) denote the changeover cost from product \( a \) to product \( b \). This cost can be calculated as follows:

\[
c_{ab} = d_{ab} + (l_{ab} \cdot \frac{165.64 + 13.84}{60^2 \cdot 2})
\] (3.2)

Let \( J_l \) denote the ordered sequence of jobs that is performed on line \( l \) and let \( p_j \) denote the product that is produced during job \( j \). Now we can calculate the changeover cost per line:

\[
c_l = \sum_{i=1}^{\mid J_l \mid-1} c_{p_ip_{i+1}}
\] (3.3)

### 3.1.2 Feeder preparation cost function

At feeder preparation, tapes are placed in feeders so that they can later be placed into the Fuji pick and place machines. Also for the feeders that are no longer needed, tapes are removed so that they can be moved back to the warehouse. In this section a cost function for the wage paid to the operators of the feeder preparation subdivision for a given line is described.

As described in Section 1.1.3 at the feeder preparation subdivision feeders are staged so that they can be reused without a transfer to warehouse. This means that for some components no feeder has to be prepared, it only has to be collected from the staging area. Also not all feeders returned from the SMD production lines have to be disassembled. A manual estimation is made by the team leader of the feeder preparation subdivision that indicates that less than 8% of the tapes returned from the SMD production lines remain in staging. Since including the described staging step into the cost function introduces an inter-line dependency on the evaluation of the cost function, it is decided to leave it out.

For the cost function of feeder preparations we use a similar approach as for the cost function of the changeovers on the SMD production line. Also a number of actions is calculated and multiplied by the costs per action. Unfortunately, the number of actions that are needed for the schedule on a line are harder to calculate than the number of changeover actions. This is because feeders stay at a SMD production line if they are used for multiple orders. Note that to keep our model simple, we do not consider multiple feeder sizes. In other words, we consider a single number of actions per line and a single average time per feeder.

On average components are returned from the SMD production line when they are not used for the next eight orders scheduled on that line. Therefore, to calculate the number of components that pass feeder preparation for a single SMD production line, a function must be evaluated that is calculated
over all jobs on this line. Calculating a delta on the evaluation of a prior, similar solution is unfeasible so it has to be completely recalculated each time the schedule of an SMD production line changes.

An efficient way to calculate the number of binding actions per line is to create a matrix of zeros with two dimensions: the first being the job index of the jobs scheduled on the concerning line and the second other being the component index of all components used in these jobs. We call this matrix \( A \). If a component is used by the job, the corresponding matrix entry is set to one. Using this matrix the number of binding actions can be calculated. This is done by running over each components and increasing a counter each time a job uses this component and not uses it for the next eight jobs. Let \( b_l \) denote the number of feeders that have to be binded in for line \( l \). The following equation gives a formula for \( b_l \):

\[
b_l = \sum_{i=1}^{\left| \mathbb{E}_1 \cup \mathbb{E}_2 \cup \ldots \cup \mathbb{E}_n \right|} \sum_{j=1}^{\left| \mathbb{C}_l \right|} A(i,j) \cdot \left( 1 - \frac{\sum_{k=j-7}^{j-1} A(i,k)}{7} \right)
\]  

(3.4)

Now that we have the number of tapes that have to be placed in a feeder, we in addition need the average time it takes to bind a tape in a feeder and to remove a tape from a feeder. The actions performed on feeders are tracked in an in-house developed software system, called the logistic execution application (LEA). An analysis of the data traced by this application can be found in Appendix A.3. From this analysis we obtain that binding a tape in a feeder takes 62.94 seconds on average and removing a tape from a feeder takes 51.17 seconds on average. A significant difference between this analysis and the analysis used in Section 3.1.1 is that this analysis uses an all inclusive interval, meaning that all spent time is included. Let \( p_l \) denote as the feeder preparation cost for line \( l \) of a single schedule. Equation (3.5) gives the feeder preparation cost function, based on the matrix calculation, the referred analysis and the fact that 8\% of the feeders can be ignored.

\[
p_l = 0.92 \cdot \left( b_l \cdot \frac{62.94 + 51.17}{60^2} \right)
\]  

(3.5)

### 3.1.3 Warehouse cost function

Tapes containing the required components for production must be picked if they are not yet at the feeder preparation subdivision or at one of the production lines. Similarly, tapes are returned to warehouse when they are no longer needed at the production lines, after they are removed from their feeders at feeder preparation subdivision.

Similar to the approach for the cost function of the changeovers on the SMD production line and the feeder preparation costs, we use a multiplication of the number of actions in a schedule, the average required time and the cost per time unit. The cost function can be derived with the same approach as used for the feeder preparation cost function and it has a similar structure.

On average components are processed by warehouse when they are not used for the next nine orders. The number of warehouse picks and storage transfers can be calculated in the same manner as the number of feeder bindings at feeder preparation subdivision. The same matrix \( A \) can be used, and we apply a slightly different function to it. Lets denote \( m_l \) as the number of warehouse picks that need to be performed. This can be calculated from \( A \) as follows:

\[
m_l = \sum_{i=1}^{\left| \mathbb{E}_1 \cup \mathbb{E}_2 \cup \ldots \cup \mathbb{E}_n \right|} \sum_{j=1}^{\left| \mathbb{C}_l \right|} A(i,j) \cdot \left( 1 - \frac{\sum_{k=j-8}^{j-1} A(i,k)}{8} \right)
\]  

(3.6)

Again we have the number of required actions; this time, these are warehouse pick actions and stock transfers. We also need a timespan per action to calculate the cost in the cost function. We use the average picking time and the stock transfer time for this.

Each occurrence of a pick action is tracked in a deployment of the SAP R/3 Enterprise information system. Here tapes are picked per order where each order is handled by one warehouse employee. Using these data the intervals between pick actions can be calculated and yield a good indication for
the cost of warehouse. From the analysis described in Appendix A.4 it can be found that a pick from stock takes 15.28 seconds on average and a stock transfer takes 22.08 seconds on average.

Note that approximately 42% of the tapes is transferred to stock. It may seem that this has to be included in the cost function, but the tapes that do not return to stock are always replaced by another tape that has been picked since. Tapes are almost never consumed by an order in full, and therefore this percentage is left out of the cost function. In Equation (3.7) these definitions are used to construct a cost function \( w_l \) for the warehouse cost for line \( l \):

\[
w_l = m_t \cdot \frac{15.28 + 22.08}{60^2}
\]  

(3.7)

### 3.1.4 Tardiness cost function

When jobs are late it causes a lot of problems with the schedule of the consecutive divisions of the operations department. Being late can also cause missed income from our customers. The costs of tardiness is estimated by the manager of the sales division at \( \varepsilon \) euros per hour. Let \( C_i \) and \( d_i \) denote the completion time and deadline of job \( i \) in seconds, respectively. Let \( J_l \) denote the set of jobs on line \( l \). Now the tardiness cost function \( t_l \) of line \( l \) can be calculated as follows:

\[
t_l = \frac{60^2}{\sum_{i \in J_l} \max\{0, C_i - d_i\}}
\]  

(3.8)

Since calculation of the tardiness is very time consuming, we also use an approximation of the change in tardiness costs in some of the algorithms (see Chapter 5). This approximation estimates the difference in tardiness costs when a sequence of jobs is shifted in time. For instance, this can be used when a single job is removed from or added to a sequence of scheduled jobs. It is calculated based on the number of successive jobs that are tardy. Let \( t_i \) be the number of tardy jobs after job \( i \), now we can calculate the delta in tardiness costs \( d_i \), when increasing the start time of job \( i \), and all successive jobs on the same machine, by \( s \) seconds as follows:

\[
d_i = \frac{60^2}{t_i} \cdot t_i \cdot s
\]

Note that the approximated change in tardiness cost is designed for small changes in the schedule, in other words, \( s \) should be small.

Note that all problem instances have many similar deadlines. This is because a goal is to fit everything within a specific window during each scheduling iteration. Alternatively, one could also include the makespan in the cost function and only include the tardiness of some jobs in the cost function. Furthermore clusters of jobs can be created for which each cluster has its own deadline. This last model is most realistic, since this is how the planning departments prepares the buckets of jobs before they can be scheduled.

Since the model becomes pretty complicated and it is unclear if we will always keep scheduling in a manner where clusters of jobs have the same deadline, we decided to introduce an individual deadline for each job. Therefore the tardiness of all jobs is included in the cost function.

### 3.2 Final cost function

The above described partial cost functions can easily be merged into a single total cost function. In this cost function all prior mentioned non optimizable costs are ignored. Furthermore, it does not take the staging of tapes, that occurs at the feeder preparation subdivision, into account. This is because taking it into consideration introduces an inter-line dependency on the cost function calculation. Including this inter-line dependency in the cost function would make an evaluation of the cost function significantly more expensive. The choice to ignore this aspect can be justified, since the staging cost spare is expected to be not significant.
Let \( \alpha \in \mathbb{R}_{\geq 0}, \beta \in \mathbb{R}_{\geq 0} \) and \( \gamma \in \mathbb{R}_{\geq 0} \) be weights that are introduced to be able to balance the factory needs based on the needs of the management team or the planning division when updating the schedule. Let \( c = c(s) \) denote the total cost of schedule \( s \). Let \( L_s \) denote the set of lines in schedule \( s \). Note that all other used variables are explained above in Equation (3.3), Equation (3.5) and Equation (3.7). The cost function for a whole schedule is given by:

\[
c = \sum_{l \in L_s} c_l + \alpha \cdot p_l + \beta \cdot w_l + \gamma \cdot l_l
\]

### 3.3 Evaluation of the cost function

All parts of the cost function can be evaluated individually. Furthermore, if the costs of certain individual lines of a schedule remain the same, then the cost function can be evaluated over the changed lines and the total cost of a schedule can be efficiently deduced. Furthermore, as shown in Chapter 5, the difference in changeover costs and the approximate difference in tardiness costs can be calculated faster. This tardiness approximation requires a calculation over the current solution of \( O(n) \) time. Using the matrix, the local difference between the changeover cost on the SMD production line of two schedules can be queried in \( O(1) \) time and the changeover cost of a schedule can be calculated in \( O(n) \) time.

Let \( n \) be the number of jobs, \( m \) the number of machines, \( n_p \) the number of distinct products in the schedule and \( o_l \) the number of distinct components on line \( l \). The following time complexities are relevant:

- **Changeover cost function**: a two dimensional matrix is calculated prior to running the algorithms. This matrix has size \( n_p \times n_p \times 2 \) and contains the changeover cost and time. This is done so that the implemented algorithms can retrieve a delta of the local differences. The creation of this matrix has to be done only once per problem instance and is calculated in two stages. The first stage selects the relevant BOM components from all BOM components, it has time complexity \( O(n_p \cdot o_l) \). The second stage builds the described changeover cost and time matrix. This matrix is build in \( O(n_p^2) \) time. Using the matrix, the local difference between the changeover cost of the schedule can be evaluated in \( O(1) \) time and the changeover cost of a schedule can be calculated in \( O(n) \) time.

- **Feeder preparation cost function**: the feeder preparation cost of the schedule can be evaluated per machine. Since the components usage of all products is known, it can be selected by running over the schedules once per component. Therefore this partial cost function can be evaluated in \( O(o_l \cdot n) \) time, and after this the number of feeder actions can be evaluated over this set.

- **Warehouse cost function**: the time complexity of the warehouse cost function is the equal to the time complexity of the feeder preparation cost function, given their identical structure.

- **Tardiness cost function**: when changes are made to a schedule, the tardiness of all changed jobs and the tardiness of all jobs subsequent to changed jobs has to be reevaluated. This reevaluation can be calculated in \( O(n) \) time. As mentioned above, some of the implemented algorithms use an approximation of the difference in tardiness costs which can be calculated faster. This tardiness approximation requires a calculation over the current solution of \( O(n) \) time where after the approximated difference in tardiness costs of all candidate solutions can be calculated in \( O(1) \) time.

Using these evaluation times of each part of the cost function, the evaluation time of the entire cost function can be determined: \( O(m \cdot (n_p^2 + o_l \cdot n)) \). The dominant part of the complexity will depend on the problem instance. Since \( n_p \leq n \) and for all used test cases \( o_l \gg n \), for these experiments the time complexity \( O(m \cdot o_l \cdot n) \) can be taken. This time complexity holds for each full evaluation of a schedule, the difference in changeover costs and the approximate difference in tardiness costs can be calculated more quickly.
Chapter 4

Exact optimization methods

In this chapter the exact optimization methods that are applicable on the described scheduling problem are discussed. First some known available exact optimization methods are discussed. In this discussion, references are given to the literature wherein these methods are first presented and/or applied. Secondly the models that we developed for exact optimization methods are presented. These models are also created to describe the practical problem more concretely. Finally the time complexity of our scheduling problem is analyzed.

The performed research described in this chapter is preceded by a pre-study wherein an overview is created of the available algorithms that may be applicable to our practical problem. This pre-study can be found in Appendix D.

In this chapter we refer to literature and create models that align with this literature. Since for the scheduling problems that can be found in the literature it is best practice to denote scheduling problems in a manner where job are to be scheduled on machines, we refer to each SMD production line as a machine and refer to each production order as a job. We use this notation from here on.

Note that in this chapter the cost function developed in Chapter 3 is considered as a basis for the development of models for the concerning scheduling problem. The models presented use this cost function as an optimization target. Apart from a cost model, also constraint exist that must be included in the models that we develop, so that only applicable schedules are yielded. The constraint that we consider are during the development of the models are:

- **Release time**: each job can have a release time as property. If this release time is set, the job must be scheduled after this time.
- **Machine constraint**: for each job a requirement can be that it only can be processed on a subset of the machines. If this requirement exist, the job may only be scheduled on one of those machines.

In Section 4.1 the exact methods that are investigated are discussed. In Section 4.2 the developed models for these exact methods are presented and discussed.

4.1 Methods to solve the problem

This section describes the exact optimization methods that can be applied on our scheduling problem. Furthermore it presents which models are needed for those methods. First mixed integer linear programming is discussed which is a commonly used technique for the modeling problems. These models of problems can be solved my multiple available solvers. After this column generation is described, which is a newer technique and for which another kind of model is required.
4.1.1 Mixed integer linear programming (MILP)

Integer programming methods can be used to solve different types of integer programming problems. This section describes some well-known existing methods that can be used to solve this kind of problems. Furthermore, some practical applications on related problems are discussed.

Integer programming is one of the 21 problems for which Karp showed that they are at least as complex as the boolean satisfiability problem (SAT) [6]. Therefore, since SAT is NP-complete, linear integer programming is NP-hard.

The described scheduling problem can be formulated as a MILP. In Section 4.2.2, we present a mixed integer program (MIP) of this problem. From this MIP, a MILP can be derived by linearizing some of the constraints. A MILP is not given since the model becomes less understandable in this form. A derived MILP can be placed in commercial tools like Gurobi and CPLEX, which will calculate the optimal value. These commercial applications are often called ILP/MILP solvers. Most of these solvers use linear programming methods such as the simplex method as sub-routine. For these problems, these LP methods do not yield feasible solutions. Often, the branch and bound algorithm is used. This algorithm uses the found unfeasible solution as a starting point. Branch and bound has an exponential time complexity, and therefore this algorithm often runs for a long time.

In [7], a near optimal solution is presented based on Lagrangian relaxation for the theoretical scheduling problem $P|p_i; w_i|\sum T$. We here use the three-field notation, which we will introduce in more detail in Section 4.2.1. Note that this problem is also NP-hard. It generates a lower bound on the cost. This can be used to measure the level of sub-optimality. It is stated that they reach within 1% of optimal for most schedules within reasonable computation time.

Although the results of this algorithm are interesting to estimate the optimality gap for the later described heuristic algorithms, we decided not to implement this and spend the available time on the evaluation of heuristic and meta-heuristic algorithms.

4.1.2 Column generation

Column generation is an exact method that can be used to solve some larger integer programming problems. As can be seen in Section 4.2.3, a theoretical (smaller than the actual) problem can be fitted into the set partitioning formulation. This formulation constitutes a basis where column generation can be applied. As far as known, column generation can tackle the largest problem instance exactly at the moment of writing this thesis.

In the paper [8], the column generation algorithm is used on a parallel machine problem. The three-field notation of the problems that were investigated are $P|p_i; L_{\text{max}}$ and $P|r_i; \text{prec}; L_{\text{max}}$. These problems can be considered simpler than the problem that is coped with during this project. It was observed that the running time increased by the number of jobs per machine (significantly starting from 20 jobs per machine).

We estimate that it will not be possible to solve the problem exactly in reasonable time. If we consider a concise set partitioning formulation of our scheduling problem, as given in Section 4.2.3, we expect that it is possible to calculate the optimal value for a problem instance of 45 jobs within reasonable time. If we consider a more precise set partitioning formulation, that represents the same model as the given MIP in Section 4.2.2, the size of the problems that are still tractable decreases significantly. It is expected that problems of at most 15 jobs can be solved within reasonable time. If the problem instance becomes bigger, we expect the run-time to increase significantly.

Although, similar as for MILP solvers, the results of this algorithm are interesting to estimate the optimality gap for the later described heuristic algorithms, we decided not to implement this and spend the available time on the evaluation of heuristic and meta-heuristic algorithms.
4.2 Models of the problem

This section gives some formal and concrete models of the practical problem that is coped with during this graduation project.

Multiple formulations for scheduling problems can be found in literature and are useful for different purposes. Section 4.2.1 gives a three-field notation of the problem. This notation is commonly used for analysis of the time complexity of scheduling problems. Section 4.2.2 gives the problem formulated as a mixed integer program this formulation is the first step to ILP solvers such as CPLEX. Section 4.2.3 gives a set partitioning formulation of the problem. This formulation is used to apply column generation techniques.

4.2.1 Three-field notation

The three-field notation \((\alpha/\beta/\gamma)\) is a commonly used notation to describe theoretical scheduling problems. It consists of the three fields \(\alpha\), \(\beta\) and \(\gamma\), where \(\alpha\) describes the machine environment, \(\beta\) describes the characteristics of the jobs and \(\gamma\) describes the objective functions. More information about three-field notation can be found in [1].

First we describe the machine environment \((\alpha = \text{MPM})\). Let \(\text{MPM}\) denote a multi-purpose parallel machine environment. This is the type of machine environment we consider in this graduation project. For a multi-purpose machine model there exists a set \(\mu_{im} \subseteq \mathcal{M}\) for each job \(i \in \mathcal{J}\) indicating that the job must be processed on one of those machines.

Secondly we describe the characteristics of the jobs \((\beta = r_i; s_{fg})\). Let \(r_i \in \mathbb{R}_{\geq 0}\) denote that for each job \(i \in \mathcal{J}\) a release time \(r_i\) exists. All jobs must start after their release time. Let \(s_{fg} \in \mathbb{R}_{\geq 0}\) define a sequence dependent changeover time between job \(f \in \mathcal{J}\) and \(g \in \mathcal{J}\). This changeover time indicates that if job \(g \in \mathcal{J}\) it the consecutive job directly after job \(f \in \mathcal{J}\) on the same machine, it can only start \(s_{fg}\) time units after job \(f\) is completed.

At last we describe the objective functions \((\gamma = \sum c_{fg}; \sum p_m; \sum w_m; \sum T_i)\). Each of these objectives represents one of the partial cost functions described in more detail in Chapter 3. Let \(c_{fg} \in \mathbb{R}_{\geq 0}\) defines the cost of a changeover between job \(f\) and \(g\). Note that, as described in Section 3.1.1, this changeover cost is not linear with the changeover time. Let \(p_m \in \mathbb{R}_{\geq 0}\) be the feeder preparation cost for the jobs on machine \(m\). Let \(w_m \in \mathbb{R}_{\geq 0}\) be the warehouse cost for the jobs on machine \(m\). Let \(T_i\) denote the tardiness cost of job \(i\). Exact formulas for these objective functions can be found in Sections 3.1.2 to 3.1.4.

Using these definitions, the theoretical scheduling problem that is coped with during this graduation project can be described using the three-field notation in Equation (4.1):

\[
\text{MPM}[r_i; s_{fg}]\sum c_{fg}; \sum p_m; \sum w_m; \sum T_i
\]  

(4.1)

4.2.2 Mixed integer program (MIP)

There are numerous ways to define the problem as a mixed integer program. In this section, it is attempted to write down a simple formulation of the problem in the form of a MIP. From this mixed integer program a mixed integer linear program (MILP) can be reduced. We present the problem as a MIP, because this form is easier to read. If required a MILP can be derived from this MIP in the future.

Let \(\mathcal{J}\) be the set of jobs that have to be scheduled. For each job \(j_i \in \mathcal{J}\) a release date \(r_i \in \mathbb{R}_{\geq 0}\), process time \(p_i \in \mathbb{R}_{\geq 0}\) and deadline \(d_i \in \mathbb{R}_{\geq 0}\) are provided. Furthermore for each combination of jobs \(j_i\) and \(j_j\) the changeover time \(d_{ij} \in \mathbb{R}_{\geq 0}\) and changeover cost \(c_{ij} \in \mathbb{R}_{\geq 0}\) are known. Let product type \(a\) be produced by job \(i\) and product type \(b\) be produced by job \(j\), using these definitions the changeover time and cost can be calculated using Equations (3.1) and (3.2).

Let \(\mathcal{M}\) be the set of machines to schedule on. Since some jobs can only be processed on a subset of \(\mathcal{M}\), we need a given constraint. Let \(\mu_{im} \in \{0, 1\}\) denote if job \(i\) can be processed on machine \(m\). \(\mu_{im} = 1\) if job \(i\) may be processed on machine \(m\) and \(\mu_{im} = 0\) otherwise.

To create a valid schedule, for each job, a start time and machine assignments must be picked. Furthermore, for each machine, an ordering must be selected. This is because the changeover
time, changeover cost, warehouse cost, feeder preparation costs and tardiness costs depend on this
ordering. Let $s_i \in \mathbb{R}_{\geq 0}$ denote the start time of job $i \in J$, $a_{im} \in \{0, 1\}$ denote the assignment of job $i \in J$
to machine $m \in M$ and $\text{succ}_{ij} \in \{0, 1\}$ denote that job $j \in J$ is the successor of job $i \in J$.

Changeovers must be performed between two jobs without any overlay in time. Therefore $s_i + p_i + \gamma_{ij} = s_j$
must hold for all consecutive jobs on the same machine in $J$.

A part of the minimization target is the tardiness of jobs. Let $T_i \in \mathbb{R}_{\geq 0}$ denote the tardiness of job $j_i \in J$. The
 tardiness can be calculated using the following formula: $T_i = \max \{0, s_i + p_i - d_i\}$ holds for all jobs in $J$. A cost for the tardiness of the jobs on a machine can be calculated using the formula $f_t$ given in
Equation (3.8). This formula for this tardiness is incorporated in the minimization target.

Let $p_m$ be the feeder preparation cost for machine $m$. A formula for this cost can be found in
Equation (3.5). Let $w_m$ be the warehouse cost for machine $m$. For this cost a formula can be found in
Equation (3.7).

To be able to evaluate the minimization target matrix $A$ must be known for each machine in a schedule.
This is because this matrix is used in Equations (3.4) and (3.6) which are used by Equations (3.5)
and (3.7). Let $b_ic$ denote that component $c$ is used by job $i$. Matrix $A$ can be derived from the output
variable $\text{succ}_{ij}$ and $a_{im}$ and the input variable $b_ic$.

This gives us the following optimization goal:

\[
\begin{align*}
\text{minimize} & \quad \sum_{i \in J} \sum_{j \in J} \text{succ}_{ij} \cdot c_{ij} + \sum_{m \in M} (\alpha \cdot p_m + \beta \cdot w_m + \gamma \cdot t_m) \\
\text{subject to} & \quad \sum_{j \in J} \text{succ}_{ij} \leq 1, \quad \forall i \in J \quad (4.2a) \\
& \quad \sum_{j \in J} a_{im} \leq 1, \quad \forall i \in J \quad (4.2b) \\
& \quad (\sum_{i \in J} a_{im}) - 1 = \sum_{i \in J} \sum_{j \in J} a_{im} \cdot \text{succ}_{ij}, \quad \forall m \in M \quad (4.2c) \\
& \quad \sum_{m \in M} a_{im} = 1, \quad \forall i \in J \quad (4.2d) \\
& \quad a_{im} = a_{jm} \lor \text{succ}_{ij} = 0, \quad \forall i, j \in J, m \in M \quad (4.2e) \\
& \quad \text{succ}_{ij} = 0 \lor s_i + p_i + d_{ij} = s_j, \quad \forall i, j \in J \quad (4.2f) \\
& \quad s_i \geq r_i, \quad \forall i \in J \quad (4.2g) \\
& \quad \mu_{im} = 1 \lor a_{im} = 0, \quad \forall i \in J, m \in M \quad (4.2h) \\
& \quad a_{im} \in \{0, 1\}, \quad \forall i \in J, m \in M \\
& \quad \text{succ}_{ij} \in \{0, 1\}, \quad \forall i, j \in J
\end{align*}
\]

Note that constraints 4.2c, 4.2e and 4.2f are not linear, but can be rewritten to obtain linear constraints.
We give a short motivation for some of the defined constraints:

- **Job sequences:** Constraints 4.2a to 4.2c ensure that there is at most one sequence of jobs per machine. Constraint 4.2a ensures that each job has at most one successor. Constraint 4.2b ensures that each job has at most one predecessor. Constraint 4.2c ensures that for each machine all pairs are connected.

- **Assignment:** Constraint 4.2d ensure that all jobs are assigned to exactly one machine.

- **Succession:** Constraint 4.2e ensure that jobs can only be successive if they are assigned to
the same machine.

- **Start time:** Constraints 4.2f and 4.2g ensure that the start times of all jobs are feasible. Constraint 4.2f ensures that the start time of a successive job is the exact time the changeover from the preceding job finishes. Constraint 4.2g ensures that all start times of all jobs are later than or equal to their release times.

- **Machine constraint:** Constraint 4.2h ensures that all jobs are only assigned to machines where they can be processed.
4.2.3 Set partitioning formulation

This section gives a set partitioning formulation that we created for our scheduling problem. It denotes a more simple cost model since the cost model described in Chapter 3 was not yet developed when this method was investigated.

This formulation is aimed to provide the possibility of solving a small problem instance using the column generation method that is discussed in more detail in Section 4.1.2. This method is expected to be more efficient than the common ILP methods (described in Section 4.1.1).

Let $S$ be an index set of schedules. An index set is a set of indexes where each index labels an element of another set. Each schedule $s \in S$ denotes a set of jobs which are ordered and for which all jobs meet their release date. Let $J_s$ be the set of jobs in schedule $s \in S$. Let $m \in \mathbb{R}_{>0}$ be the number of available machines.

Let $d_i \in \mathbb{R}_{\geq 0}$ denote the deadline for job $i \in J_s$ and let $C_i \in \mathbb{R}_{\geq 0}$ denote the completion time of job $i \in J_s$. The tardiness $T_i \in \mathbb{R}_+$ of a job $i \in J_s$ is fully derivable from these two properties, therefore $T_i = \max\{0, C_i - d_i\}$.

Let $c_s \in \mathbb{R}_{\geq 0}$ denote the calculated changeover time of schedule $s \in S$. Furthermore let $w_s \in \mathbb{R}_{\geq 0}$ denote the calculated changeover work effort of schedule $s \in S$. Let $T_s \in \mathbb{R}_{\geq 0}$ denote the calculated total tardiness of schedule $s \in S$, this results in the following equalities:

\[
c_s = \sum_{i=1}^{\left|J_s\right|-1} c_{i,i+1} \quad w_s = \sum_{i=1}^{\left|J_s\right|-1} w_{i,i+1} \quad T_s = \sum_{i=1}^{\left|J_s\right|} T_i
\]

Let $a_{s,i} \in \{0, 1\}$ be a binary decision variable indicating if job $i$ belongs to schedule $s$, $a_{s,i}$ is equal to 1 if and only if job $i$ belongs to schedule $s$. Let $\lambda_s \in \{0, 1\}$ be another binary decision value indicating if schedule $s$ belongs to the optimal solution. $\lambda_s$ is equal to 1 if and only if schedule $s$ belongs to the optimal solution.

The variables $\alpha \in \mathbb{R}_{\geq 0}$ and $\beta \in \mathbb{R}_{\geq 0}$ are provided weights that are introduced to balance the cost functions. With these definitions the scheduling problem can be represented as a set partitioning problem with some additional constraints:

\[
\text{minimize} \quad \sum_{s \in S} (\alpha \cdot c_s + \beta \cdot w_s + T_s) \cdot \lambda_s \\
\text{subject to} \quad \sum_{s \in S} a_{s,i} \cdot \lambda_s = 1, \quad \forall i \in J \quad (4.3a) \\
\sum_{s \in S} \lambda_s \leq m, \quad (4.3b) \\
\lambda_s \in \{0, 1\}, \quad \forall s \in S, \forall i \in J
\]

Constraint 4.3a defines that all jobs must be part of one of the selected schedules. Constraint 4.3b defines that at most one schedule per machine is selected.

4.3 Analysis of time complexity

In this section an analysis of the time complexity of the described scheduling problem is given. In this analysis we find that the described problem in Section 4.2.1 is NP-hard. We use the three-field notation for this proof since this is the most concise formulation that can be used for our- and intermediate scheduling problems.

As can be found in [9] the scheduling problem $1|r_i|\sum C_i$ is strongly NP-hard. In this section we reduce this problem to our problem which will show that our problem is harder or as hard as this problem. Therefore our problem is also in the class NP-hard.
The proof is based on five problems \( X_1, \ldots, X_5 \) that can be reduced in the given order. The five scheduling problems that are used for this proof are:

\[
\begin{align*}
X_1 & : 1|r_i|\sum C_i \\
X_2 & : 1|r_i|\sum T_i \\
X_3 & : P|r_i|\sum T_i \\
X_4 & : P|r_i|s_{fg}|\sum c_{fg}i|\sum p_m|\sum w_m|\sum T_i \\
X_5 & : MPM|r_i|s_{fg}|\sum c_{fg}i|\sum p_m|\sum w_m|\sum T_i \\
\end{align*}
\]

The tardiness, defined by \( T_i \), is based on the completion time and job deadline in the following manner:

\[ T_i = \max \{0, C_i - d_i\} \].

For each problem instance from \( X_1 \) in combination with \( d_i = 0 \) for all jobs, \( X_2 \) can be used to solve it. Therefore, \( X_1 \) can be reduced to \( X_2 \) and thus \( X_1 \leq_m^p X_2 \).

The machine environment \( \alpha = P \) used in problem \( X_3 \) indicates that there must be scheduled on \( m \) identical parallel machines. All problem instances from \( X_2 \) in combination with \( m = 1 \) can be solved by \( X_2 \), therefore \( X_2 \) can be reduced to \( X_3 \) and thus \( X_2 \leq_m^p X_3 \).

The duo job dependent variable \( s_{fg} \) used in problem \( X_4 \) indicates that a sequence dependent changeover time must be considered. The variables \( c_{fg}, p_m \) and \( w_m \) presents sequence dependent objectives that are not considered in the schedule. Therefore these objectives can be ignored. All problem instances from \( X_3 \) in combination with \( s_{fg} = 0 \) for all jobs \( f \) and \( g \) can be solved by \( X_4 \). Therefore, \( X_3 \) can be reduced to \( X_4 \) and thus \( X_3 \leq_m^p X_4 \).

The difference between the \( MPM \) and \( P \) machine environment not jobs can be constraint to a subset of the machines, meaning that they can only be processed on those machines. All problem instances from \( X_4 \) in combination with \( \mu_{im} = M \) for all jobs \( i \) can be solved by \( X_5 \). Therefore, \( X_4 \) can be reduced to \( X_5 \) and thus \( X_4 \leq_m^p X_5 \).

From these reductions, it can be concluded that \( X_1 \) can be reduced to \( X_5 \), thus \( X_1 \leq_m^p X_5 \) and thus the described scheduling problem \( X_5 \) is also in the class NP-hard.

If we consider a decision problem that is derived from the optimization problem, it is left to check if it is also in the subclass NP-complete. All problems that are in the classes NP and NP-hard are in the class NP-complete. Therefore it is left to check if the described decision problem is in the class NP.

For a decision problems, NP is the class of all problems that can be solved non-deterministically by a turing machine. A first property that a decision problems should have to verify this is, if a verification algorithm is known that proves that a given instance is valid. Furthermore it must be feasible to calculate this verification algorithms in polynomial time. If this is the case, the decision problem is part of the class NP.

To verify this a decision problem should be derived from the given optimization problem. Let \( o \) define a given objective for the cost function. Let the decision problem be: is the cost of a given schedule less than \( o \). For each solution this can be verified in polynomial time. Therefore such a verification algorithm is known for this decision problem and therefore it is in NP-complete.
Chapter 5

Heuristic optimization algorithms

This chapter describes the classes of heuristic optimization algorithms that are investigated during this graduation project. It substantiates how different variants of heuristic algorithms are implemented and puts these in classes of heuristic algorithms.

The performed research described in this chapter is preceded by a pre-study wherein an overview is created of the available algorithms that may be applicable to our practical problem. This pre-study can be found in Appendix D.

All described heuristic algorithms evaluate the cost function described in Chapter 3. Each implementation complies with the constraints specified in the MIP developed during this project. This MIP can be found in Section 4.2.2. The used test and validation sets are described in Chapter 6. The results of optimizing the cost function in experiments using the heuristic algorithms described in this chapter are discussed later on, in Chapter 7.

We first describe the methods that are used to compose schedules from a bucket of jobs. This is presented in Section 5.1. Secondly, in Section 5.2, we discuss the different possible neighborhood structures. In Section 5.3 we present how improvement in the costs can be visualized in different manners. In Section 5.4, we give the different variants of local search methods. Note that multiple of the latter described meta-heuristic algorithms use concepts of the described local search methods. After this the classes of meta-heuristic algorithms, from which certain representatives are implemented, are discussed. These classes are tabu search, simulated annealing and genetic algorithms, described in Sections 5.5 to 5.7 respectively.

5.1 Schedule generation methods

All heuristic algorithms, described later on in this chapter, need one or more feasible schedules during their initialization. Furthermore some algorithms need input schedules while they are running. This is for example required when the population of a genetic algorithm has to be filled up (see Section 5.7).

This section describes which methods are used to generate schedules:

- Production schedule: when we test with the created validation set, we use schedules that are created based on historical schedules from the SMD PCBA manufacturing division. This means that we take a part of a schedule that was manually created by the shop floor planner. From this schedule we reuse sequences of orders that where produced on the lines. We take these sequences from the same starting point on different lines. From the selected jobs certain jobs at the start are considered constants in the schedule. Manually, realistic release times and deadlines are added to the jobs in these test cases.

- Random schedule: we implemented one stochastic method. This method starts with a schedule containing only the constant jobs, where after each job in the bucket is placed one-by-one. In each iteration, first one of the machines, for which the machine makespan is the lowest, is selected. For this machine a random job from a bucket of candidate jobs is selected. Jobs are
only considered a candidate if the makespan on the machine is higher than the release time of the concerning job. A pseudo-code representation of this method can be found in Algorithm 1 presented in Appendix B.

- **Greedy schedule**: one greedy method is implemented that creates a schedule while being greedy on the changeover cost, as described in Section 3.1.1. This method also first selects the machine for which the makespan is the lowest. Then it selects the candidate solution for which the changeover cost from the last job on the selected machine is the lowest. The job with the lowest changeover cost is placed in the schedule. After this the procedure is repeated. A pseudo-code representation of this method can be found in Algorithm 2 presented in Appendix B.

## 5.2 Neighborhood structure

Most of the implemented heuristic algorithms use a method where they evaluate one or more of the solutions in the neighborhood of a single solution. A neighborhood of a solution is a set of solutions that can be created by performing a small modification to the source solution. The small modification that is used defines the structure of the neighborhood. In this section we describe the neighborhood structures that are used by the implemented heuristic algorithms.

In general heuristic algorithms are used for problems for which evaluating all solutions in the search space is not tractable. Rather than evaluating the whole search space, these heuristic algorithms then evaluate a small part of the search space. To determine the part of the search space that is to be evaluated often a perspective from a single solution is used.

When looking from the perspective of a single feasible solution, changes/modifications can be made which will result in other solutions. These solutions are often called the neighbors with respect to the source solution. A neighborhood structure is a structure that defines how changes are made starting from this solution. We define the neighborhood structure $\mathcal{N}(s)$ as a function of schedule $s \in \mathcal{S}$ where $\mathcal{S}$ is the search space:

$$\mathcal{N} : \mathcal{S} \rightarrow \mathcal{U}_s \subset \mathcal{S} \text{ with } \mathcal{U}_s \text{ a neighborhood of } s$$

In 1996 D. Mattfeld identified characteristics that identify a good neighborhood structure [10]. These characteristics are:

- **Correlation**: a neighbor has only a small difference from its originator. For scheduling problems this normally means that the larger part of the jobs stay in place.
- **Feasibility**: a neighborhood should only consist of feasible solutions. A feasible solution satisfies all constraints that are defined for the model.
- **Improvement**: for each neighbor there should be a significant probability that it improves the objective value.
- **Size**: the number of solutions in a neighborhood should be in balance with the computation time of a solution. This should give a balance between the neighborhood size and the number of iterations.
- **Connectivity**: a finite sequence of moves leading to a global optimum must exist. In our case, a finite sequence to the global minimum must exist.

We expect the performance of most heuristic algorithms to change significantly depending on the neighborhood structure. How the neighborhood of a solution can be selected, bounded and evaluated is studied in very many papers. For example, J. Kuhpfahl recently performed a study on local search neighborhoods for the job shop scheduling problem with a weighted tardiness objective [11]. Li et al. performed research on a similar model with a different fitness function. They consider two neighborhood structures, namely job swap and job jump. In 2004 Ima et al. [12] proposed a neighborhood structure specifically for job shop scheduling minimizing makespan. A fundamental
concept in this paper is that of alternating neighborhood structures to escape from local minima. A. Misevicius proposed a combination of 2, 3 and 4-opt [13] for solving the traveling salesman problem.

5.2.1 Specific neighborhood structures

Below we list some of the most commonly applied neighborhood structures for scheduling problems. Also a few more general neighborhood structures are discussed. In theoretical research, these neighborhood structures are often applied to the Traveling Salesman Problem (TSP). We also briefly explain why certain neighborhood structures are not implemented.

We give a short description of the neighborhood structures that are used by the heuristic algorithms implemented during this graduation project:

- **job jump**: job jump forms a neighborhood that can be constructed by removing a job from the candidate solution and placing it somewhere else on the schedule. If performing a job jump will result in a feasible schedule, this schedule an element of the neighborhood of the candidate solution. In Figure 5.1 an example is given of how such a neighbor is constructed. A known disadvantage of this neighborhood structure is that it may get stuck in a local minimum when lines are perfectly balanced and deadlines are tight. As described in Chapter 6 most jobs have the same deadline. Since in our case not so many deadlines are tight, we do not expect that stagnation in local minima because of this reason often occurs.

- **job swap**: the neighborhood structure created by swapping jobs is also referred to as job interchange. A neighbor can be constructed by swapping the location of two jobs in the schedule. If performing a job swap will result in a feasible schedule, this schedule is an element in the neighborhood of the candidate solution. In Figure 5.2 an example of how such a neighbor is created is given. When a schedule is very tight on deadlines, using this neighborhood structure may still provide better neighbors while using the job jump neighborhood structure may already evaluate the candidate as a local minimum. A known disadvantage is that the number of jobs on each machine always stays the same which limits the reachable part of the solution space significantly.

- **2-opt**: 2-opt was first proposed by G. Croes in 1958 [5]. It was designed to create a neighborhood structure for the TSP. Since our scheduling problem can be seen as a multi-path cover of a close to fully connected graph, 2-opt can also be applied to it. 2-opt splits two edges in the graph and connects the open vertices in the opposite manner. In Figure 5.3 an example is given on how this will influence a schedule. 2-opt has been studied in many papers [1, 14, 15] and is also described in an operations research book by Ho et al. which treats optimization of PCBA production planning [16].

- **3-opt**: similar to 2-opt, 3-opt can be applied on all graph problems and is extensively applied to the TSP [13, 17, 18]. It is a specification of the more abstract neighborhood structure k-opt. It is applied by deleting three edges in a graph cover and reconnecting them in a different manner. To scope this graduation project, 2 and 3-opt are implemented and higher k values are not considered.

Figure 5.1: A job jump mutation performed on job \( J_2 \) in an example schedule.

In the literature many other neighborhood structures are treated. We discuss a few well known examples, which, however, are not implemented during this graduation project:
- **Block jump**: a block jump neighborhood structure is a neighborhood structure where neighbors are constructed by removing a block of jobs from the schedule and placing it somewhere else. This neighborhood structure is described and treated in multiple papers [11, 19, 20]. As described in Chapter 1 and Chapter 6 the expected problem instances are of a small or medium-size. Since this neighborhood structure is targeting larger sized problems we decided not to implement it. We expect that when this neighborhood structure would be implemented, after hyper-parameter tuning, the block size would be set to one job. Furthermore we expect that moving blocks of more than one job will result in the evaluation of more unfeasible solutions.

- **Block swap**: similar to the block jump mutation being an extension on the job jump mutation, the block swap mutation is an extension of the job swap mutation. A neighbor is constructed by swapping two blocks of jobs in the schedule. We decided to not implement this with the same arguments as for the block jump neighborhood structure.

![Figure 5.2: A swap mutation performed on job $J_2$ and $J_5$ in an example schedule.](image)

![Figure 5.3: A 2-opt mutation performed in an example schedule.](image)

### 5.2.2 Neighborhood size

In many studied papers a neighborhood structure is bounded by a neighborhood size. This quantity limits the number of solutions in the neighborhood of the present candidate solution. The goal of this bound is to limit the number of solutions that have to be evaluated in each neighborhood evaluation. This is done to decrease the computation time per iteration for heuristic algorithms. For example, to increase the number of iterations in the optimization time window or because the evaluation of a single neighborhood is not tractable without a size constraint.

One of the disadvantages of the use of a neighborhood size is that methods based on it may reach relatively poor local minima. Another minor disadvantage is that a function determining which solutions are in the neighborhood may increase the complexity of the implementation and therefore may introduce a small increase of overhead in computation time per iteration.

The main advantage of the use of a neighborhood size is that it decreases the computation time of an iteration, since fewer solutions have to be evaluated.

In the setting of our problem, we also use approximate neighborhood evaluations (see Chapter 3). In that case the evaluation time of an iteration is very small which introduces the possibility of working with larger neighborhoods.
When the problem size is small, bounding the neighborhood size may be pure overhead since the problem size already limits the time dimension. For larger schedules, for example the schedules as described in Section 1.2, the neighborhood size is expected to be essential for heuristic algorithms that exactly evaluate the solutions in the neighborhood. This is because otherwise an exact evaluation of the cost function is expected to be too costly already for a single iteration.

The neighborhood of a candidate solution can be bound in multiple dimensions. For scheduling problems the dimensions machine and time are often used. We first look at the machine dimension. Creating a constraint for this dimension means that jobs may at most move a certain number of machines. Since the machine constraints are on close machines (currently machine six and seven) introducing a size constraint on the machine dimension may align the neighborhood a little bit with the constraints. Since the number of machines is small and does not normally increase for larger problem instances it was decided to not implement a bound for this.

For scheduling problems bounding the neighborhood size in the time dimension can be implemented in two manners:

- The mutation method can be bounded by time. As an example, for the job jump mutation neighborhood structure this means that only candidate schedules for which the job moved at most a specific amount of time from its origin are considered in the neighborhood.
- The mutation method can be bounded by a maximum number of positions that it may shift. For example, for an algorithm that uses the job swap mutation neighborhood structure, this means that only candidate solutions are considered for which jobs are swapped that are at most a specific amount of positions apart from each other. When swapping jobs on multiple machines the job for which its center is closest on the target machine is used as reference.

We decided to implement a single method to limit the neighborhood size and picked the constraint that limits the number of positions that the mutation points are apart. We selected this method because this is expected to have less variance in the number of neighbors of candidate solutions. The risk of limiting the neighborhood size by the amount of time is that a huge amount of possible neighbors arise in places with short jobs and no possible neighbors may exist when long jobs are run on all machines during the time of the job that is considered to be mutated.

5.3 Visualization of cost change

Some heuristic algorithms walk through the search space over a sequence of solutions for which the evaluated cost of these solutions stays the same or decreases. An example of this is a local search algorithm. Other heuristic algorithms walk through the search space in a manner where they sometimes move to solutions with higher costs. Examples of this are the tabu search algorithm and simulated annealing. More details about these algorithms are given in Sections 5.5 and 5.6. For these algorithms plotting the costs of evaluated solutions over time may also increase. In this thesis we plot two types of stair plots that visualize the cost change during the performed experiments. These two types of cost change are the cost of the current solution and the cost of the incumbent solution.

An example of how these two types of change in costs is visualized is given in Figure 5.4. The cost of the incumbent solution can be derived from the cost of the current solution over time. Its cost can be calculated using the formula in Equation (5.1). Note that in Chapter 7 the cost of the incumbent solution is visualized in separate figures.

\[ f(i) = \min_{j \leq i} f(j) \]  (5.1)
5.4 Local search (LS)

The class of local search algorithms is the most basic class of heuristic algorithms that can be applied to combinatorial optimization problems. There are multiple classes of meta-heuristic algorithms that extend this class of algorithms. Examples of such algorithms are the tabu search algorithm (see Section 5.5) and simulated annealing (see Section 5.6). These classes are discussed later.

A local search algorithm without the use of a meta-heuristic algorithm will always get stuck in the first evaluated local minimum. It starts from a single solution of the solution space, evaluates all solutions within a neighborhood and moves on to a better solution. Some local search algorithms select the first fitter solution found in the neighborhood, other local search algorithms evaluate a part of the neighborhood or the whole neighborhood and move on to the fittest one. In our implementation we look at a partial neighborhood that can be composed by all mutations from the perspective of a single job or changeover and perform the mutation that results in the best feasible solution. If for all starting points, there are no fitter solutions in the partial neighborhood of the candidate solution, the best solution in the neighborhood is reached. Since there may still be better solutions in the search space, the candidate solution is a local optimum rather than a global optimum. The goal of meta-heuristic algorithms is to escape such local minima.

For the created test cases we use the greedy algorithm to construct a schedule and for the created validation cases we use the production schedule. How these schedules are constructed is described in more detail in Section 5.1. More information about the used test and validation cases can be found in Chapter 6.

Local search algorithms can be implemented in multiple manners. The main difference in the different implementations is the neighborhood structure. In the implemented local search algorithms three neighborhood structures are used. These neighborhood structures are job jump, job swap and 2-opt. For each neighborhood structure experiments are performed with and without neighborhood size constraint. These neighborhood structures and how neighborhood size constraint is applied is described in more detail in Section 5.2.

Note that with most implementations of the local search algorithms the fitness of all solutions in the neighborhood is exactly evaluated, and then the best option is picked. In the case of our problem, this requires a recalculation of a significant part of the cost function for the lines that are changed. As explained in Chapter 3, the time complexity of this calculation is significant. Therefore, also a second local search algorithm is implemented that evaluates an approximation of the cost function. Only for the neighbor with the best approximate cost, the exact cost function is calculated.
For the local search algorithm that uses an approximate evaluation of the cost function, a best instance is selected based on the changeover difference, which is the exact difference in changeover cost and the tardiness approximation. This approximation can be calculated in $O(1)$ time. After finding the neighbor with the best approximate improvement the cost function is evaluated exactly for it. If the exact fitness improves, the solution is selected as next candidate solution.

A pseudo-code representation of a local search algorithm that uses exact evaluation of the neighborhood and a pseudo-code representation of a local search algorithm using approximate evaluation are given in Algorithms 3 and 4. These algorithms can be found in Appendix B. In Figure 5.5 a flowchart of the implemented class of local search algorithms is given. In this figure the different implemented variants that can be picked for this class are indicated.

![Flowchart of the process executed by the class of local search algorithms.](image)

**5.5 Tabu search (TS)**

The tabu search algorithm is a meta-heuristic algorithm that uses the same concepts as a local search algorithm, but extends it with the use of a tabu list. This tabu list is a list of solutions that are no longer considered in any neighborhood. Furthermore, if the best solution of an evaluated neighborhood is worse in costs than the candidate solution, it is selected for the next iteration. If such a worse solution is selected, the last solution is placed in the tabu list so that it can be avoided during subsequent iterations. The tabu algorithm was proposed by Fred W. Glover [4].

When an iteration of the tabu search selects a solution with a higher cost as candidate solution, on the next iteration the algorithm will ignore the source solution since this solution will be in the tabu list. Then from the new candidate solution another decreasing path may be found.

The tabu list has a specific length. When the tabu list is full and a solution is found that should be added to the tabu list, another solution must be removed from the list. Items are removed from the tabu list using the First In First Out principle (FIFO). When the length of this list is very large or when the list can grow indefinitely, the iterations of the algorithm become longer and longer over time. This is because the comparison of the neighbors of the candidate solution with this list becomes a more time consuming job.
Using a too short tabu list may cause the algorithm to wander around in a closed loop through the search space. This can happen when at some point schedules are removed from the tabu list that are reachable by the candidate solution and will be still evaluated as best solution. Therefore it is important to pick a proper tabu list length.

For the whole class of implemented tabu search algorithms the complete neighborhood is evaluated, where after the best neighbor is selected. Note that this neighborhood is significantly bigger than the neighborhood that is selected for the class of local search algorithms.

The neighborhood structures job jump and job swap are implemented since they perform better than the k-opt neighborhood structures in the performed experiments described in Chapter 7. For each neighborhood structure experiments are performed with and without neighborhood size constraint. These neighborhood structures and how neighborhood size constraint is applied is explained in more detail in Section 5.2.

Note that similar to the the class of local search algorithms, approximate and exact neighborhood evaluation are implemented. Since we work with significantly larger neighborhood structures in this class of algorithms, the evaluation time of a neighborhood is expected to increase significantly compared to local search algorithms.

A pseudo-code representation of the tabu search algorithm is given in Algorithm 5 which can be found in Appendix B. In Figure 5.6 a flowchart of the implemented class of tabu search algorithms is given. In this figure the different implemented variants that can be picked for this class are indicated.

![Flowchart of the process executed by the class of tabu search algorithms.](image)

Figure 5.6: A flowchart of the process executed by the class of tabu search algorithms.
5.6 Simulated annealing (SA)

Simulated annealing is the first non-deterministic algorithm described in this thesis. Similar to the tabu search algorithm it is a meta-heuristic algorithm that reuses some concepts of the local search algorithm. It was proposed by Kirkpatrick et al. in 1983 [21].

Simulated annealing is a stochastic local search algorithm that changes its behavior during the traversal of the search space from stochastic to more and more deterministic. If implemented correctly, simulated annealing will in the first part of its execution evaluate solutions in a common high area of the search space, and then slowly descending to better solutions. One of the advantages of the stochastic nature of this algorithm is that it will always have a positive probability of escaping a local optimum and it will eventually always reach the global optimum. In other words, a path to the global optimum exists from every solution.

![Flowchart of the process executed by the class of simulated annealing algorithms.](image)

The simulated annealing algorithm, in each iteration evaluates a single neighbor of the candidate solution. This means that this is the first described algorithm that does not evaluate multiple neighbors within a single iteration. Therefore we only use exact evaluation of the cost function when implementing this algorithm.

To be able to select a random schedule that is a neighbor of the candidate solution we need to define a neighborhood structure. Experiments have been performed with multiple neighborhood structures. For simulated annealing we implemented the job jump, job swap, 2-opt and 3-opt neighborhood structures. These neighborhood structures are described in more detail in Section 5.2.
If the randomly selected neighbor evaluates a better cost than the candidate solution, it is selected as candidate solution for the next iteration. If the selected neighbor results in a worse cost than the candidate solution, it still has a chance to be selected for the next iteration. In this case a random between one and zero is picked that is compared to a cooling scheme. If the random number is lower than the value provided by the cooling scheme, the neighbor is selected as the candidate for the next iteration. If the random is higher than the value provided by the cooling scheme, the candidate does not change for the next iteration. The possible and implemented cooling schemes are further discussed in Section 5.6.1.

In some studied papers simulated annealing is implemented in a slightly different manner. It includes a step where the best neighbor in a neighborhood is selected. If the best solution in the neighborhood is an improvement, it is selected as the candidate solution in the next iteration. This significantly increases the computation time per iteration. Implementing simulated annealing in this manner can be tricky because the balance between exploration and exploitation needs to be maintained. Using this technique variations of the simulated annealing algorithm can be implemented for which using the described approximate cost function evaluation can be beneficial.

A pseudo-code representation of the simulated annealing algorithm is given in Algorithm 6 presented in Appendix B. In Figure 5.7 a flowchart of the implemented class of simulated annealing algorithms is given. In this figure the different implemented variants that can be picked for this class are indicated.

### 5.6.1 Cooling scheme

When during an experiment with a simulated annealing algorithm a neighbor is evaluated that results in a worse fitness than the candidate solution, it may still be picked as the candidate solution for the next iteration. Whether this solution is picked as the next candidate depends on the cooling scheme. This section describes some possible cooling schemes.

All described cooling schemes influence a temperature value $T_n$ which is gradually decreasing over time. When $T_n$ approaches zero, the implemented algorithm behaves deterministic. Then it can be compared to first acceptance hill climbing which is a local search algorithm that moves to the first found better neighbor. There are many different cooling schemes and picking the wrong cooling scheme may yield in bad results.

Stander et al. [22] and Nourani et al. [23] compared cooling schemes in 1994 and 1997 respectively. Lee et al. applied a simulated annealing cooling scheme on parallel machine scheduling with sequence dependent setup times in 1997 [24]. Kim et al. compared different SA methods on the same problem in 2002 [25]. In these papers we can find the following cooling schemes:

- **Linear scheme**: a linear scheme decreases the temperature linearly by the number of iterations multiplied by $\eta$. A linear scheme has been widely used. $\eta$ is a hyper-parameter indicating how fast the linear descent acts. Since the run-time in our experiments is always known before running, $\eta$ can be picked so that $T$ is exactly zero on the last iteration. A formula for the temperature of the linear scheme can be found in Equation (5.2).

- **Exponential scheme**: In the paper where simulated annealing is proposed by Kirkpatrick et al. in 1983 [21], an exponential cooling scheme is used. For this an initial value is picked ($T_0$). Equation (5.3) gives the formula for sequential temperatures. Often in literature $T_1/T_0$ is denoted as $\alpha$ and therefore the formula for the temperature can also be denoted relative to the last temperature: $T_n = \alpha \cdot T_{n-1}$. In the experiments performed in their paper, a geometric ratio of $T_1/T_0 = 0.9$ is used. In the experiments performed in this thesis $T_1/T_0$ is considered a hyper-parameter and is tuned per experiment.

- **Logarithmic scheme**: the logarithmic scheme for simulated annealing was first proposed by Geman et al. in 1984 [26]. A formula for the temperature of the logarithmic scheme can be found in Equation (5.4). The hyper-parameter $c$ scales the height of the logarithmic function and $d$ shifts the time dimension.

- **Asymptotic scheme**: In 1988 B. Hajek [27] introduces a variation on the work of S. Geman at al. which essentially sets $d = 1$. See Equation (5.5).
A linear scheme is easy to implement and experiment with. It is widely used and this is the first cooling scheme picked for simulated annealing. In 2002 Kim et al. investigated simulated annealing applications for parallel machine scheduling problems [25]. In their work, it can be found that the exponential scheme is used more commonly in practice and therefore this cooling scheme is also implemented. When using this cooling scheme often $\alpha$ is picked between 0.5 and 0.99. For most applications the exponential scheme has a better balance between exploitation and exploration than the linear scheme.

For the logarithmic cooling schemes B. Hajek proved that if $c$ is set beyond the best known local optima, eventually the global optimum will be reached [27]. Although this is a nice property of the cooling scheme, it is also known that due to its extremely slow decrease of temperature, these cooling schemes are very impractical [23]. Therefore we did not implement a logarithmic cooling scheme.

$$T_n = T_0 - \eta \cdot n \quad (5.2)$$
$$T_n = T_0 \cdot T_0 \quad (5.3)$$
$$T_n = \frac{c}{\log(d + n)} \quad (5.4)$$
$$T_n = \frac{c}{\log(1 + n)} \quad (5.5)$$

### 5.7 Genetic algorithm (GA)

In 1975 J. Holland proposed the genetic algorithm (GA) [28]. A genetic algorithm is an evolutionary, population based algorithm. It relies on the concept of survival of the fittest. It is constructed out an initialization function and an iterating part. The iterating part of the algorithm exists out of three operations namely selection, crossover and mutation. These operations are repeatedly and consecutively performed in the given order. Each of the operations changes the solutions in the population and can be implemented in numerous ways. Picking these operations poorly may cause intensification of the population to a subset whereof all solutions are fairly similar and therefore close to each other in the search space.

![Figure 5.8: A flowchart of the process executed by the class of genetic algorithm.](image)

A pseudo-code representation of the genetic algorithm can be found in Algorithm 7 presented in Appendix B. Since the introduction of GA extensive research has been performed. The most relevant research is the research performed for similar problems, for example, the job shop and parallel machine scheduling problems.
In 1995 D. Christian wrote a book on evolutionary search for job shop scheduling in which he describes GA for job shop schedule. In 1996 Cheng et al. wrote a tutorial survey on solving job shop scheduling problems using genetic algorithms [29]. In 1999 they continued with part two which describes some hybrid search strategies [30]. In 2011 Werner performed a similar survey [31]. In 2010 Hu et al. published a paper on GA for parallel machine scheduling with sequence dependent setup times [32]. Especially this paper describes a problem very similar to the problem coped with during this graduation project. In 2018, Adan et al. presented a Hybrid GA for parallel machine scheduling for a specific application namely semiconductor back-end production. [33].

Figure 5.8 shows a flowchart of the most common operations performed by implementations of GA’s. Hereafter, each of these operations is described individually.

5.7.1 Initialization

During initialization the initial population is created. Recently there is a trend that some good individuals are included in the initial population [33]. Therefore the picked initial population exists out of a greedy algorithm solution supplemented with random valid solutions. These schedules are created using the methods described in Section 5.1. How big the population is and why this size is selected is further described in the Chapter 7.

5.7.2 Selection methods

During the selection procedure, the elements of the population are selected that are taken to the next step. During the crossover and mutation procedures, these are extended so that the population obtains the same size again.

Selection methods for GA in general are studied in recent papers by K. Jebari [34] and Lata et al. [35]. AL-Rawashdeh et al. studied selection methods specifically for scheduling problems [36].

The most commonly known selection methods are: survivor selection, roulette wheel selection, reward based selection, stochastic universal sampling, truncation selection and tournament selection. AL-Rawashdeh et al. find that roulette wheel and tournament selection are most used in GA for solving scheduling problems. Since Hu et al. used roulette wheel selection for the problem that they describe, this selection method is implemented.

Since the fitness of each chromosome is fairly high and therefore the difference in height is not significant on the roulette wheel, first min-max normalization is applied. A formula for min-max normalization is given in Equation (5.6):

\[
c_{\text{scaled}} = \frac{c - c_{\text{min}}}{c_{\text{max}} - c_{\text{min}}}
\]  

(5.6)

The roulette wheel selection method selects a single element for a set. This method is repeated until enough solutions are selected for the next iteration. This method ‘places’ a set of values, in our case \(c_{\text{scaled}}\) for the whole population, in a roulette wheel where the relative amount of each item determines the amount of degrees this element gets on the roulette wheel. Then a random is picked that selects an angle on the roulette wheel. With this angle an item is selected. Define \(P\) as the population than the change of \(s_1 \in P\) being selected in the first iteration can be calculated using Equation (5.7):

\[
P(s_1) = \frac{c_{\text{scaled}}(s_1)}{\sum_{s \in P} c_{\text{scaled}}(s)}
\]  

(5.7)

5.7.3 Crossover methods

The crossover operation is a method that is defined as the combination of two parents to generate a new chromosome. Crossover operations are extensively studied, for example Otman et al. and W. Hameed performed studies on crossover operators for the traveling salesman problem [37, 38] and Anand et al. performed a similar study for the open shop scheduling problem [39].
Some well known crossover method are: one point, two point, linear order, cycle, position based and order based crossover. During testing with genetic algorithms one and two point were implemented. Since from literature [32] it is known that order based crossover method is commonly used for similar problems, this is also implemented. A good description of this crossover method can be found in [39]. Recently, more complex crossover operators are proposed, for example Vallanda et al. proposed a fast and a slow local search enhanced crossover operator [40].

5.7.4 Mutation methods

Mutation methods avoid convergence of the population. Most of the mutation operators are comparable with the local search neighborhood selection methods described in Section 5.2. In 2014 Soni et al. performed a general study on various mutation operations [41]. In 2012 Abdoun et al. performed a similar study, more focused on the TSP [42]. For the sake of simplicity, in the initial implemented genetic algorithm a job swap mutation is performed.

Since some results during testing indicated that the population might converge to a small subspace of the solution space a second selection method is introduced where new random schedules are added to the population instead of close variations the current population.

5.8 Overview

In this section, we give an overview of the algorithms that are implemented. Each different implementation is assigned a distinct label, to make it easier to reflect on the results in Chapter 7. Since the implemented genetic algorithms have different properties than the other classes of algorithms, they are labeled separately in Table 5.2. All other algorithms are given in Table 5.1. In total 44 algorithms from five different classes of algorithms are implemented.
Table 5.1: An overview of the implemented local search, iterated local search, tabu search and simulated annealing algorithms.

<table>
<thead>
<tr>
<th>Class</th>
<th>Neighborhood structure</th>
<th>Neighborhood constraint</th>
<th>Neighborhood evaluation</th>
<th>Cooling scheme</th>
<th>Label</th>
</tr>
</thead>
<tbody>
<tr>
<td>Local search</td>
<td>jump</td>
<td>no</td>
<td>c</td>
<td>LS</td>
<td>c jump</td>
</tr>
<tr>
<td>Local search</td>
<td>jump</td>
<td>yes</td>
<td>c</td>
<td>LS</td>
<td>c jump +</td>
</tr>
<tr>
<td>Local search</td>
<td>jump</td>
<td>no</td>
<td>c</td>
<td>LS</td>
<td>c jump</td>
</tr>
<tr>
<td>Local search</td>
<td>swap</td>
<td>yes</td>
<td>c</td>
<td>LS</td>
<td>c swap</td>
</tr>
<tr>
<td>Local search</td>
<td>swap</td>
<td>no</td>
<td>c</td>
<td>LS</td>
<td>c swap +</td>
</tr>
<tr>
<td>Local search</td>
<td>2-opt</td>
<td>no</td>
<td>c</td>
<td>LS</td>
<td>c 2-opt</td>
</tr>
<tr>
<td>Local search</td>
<td>2-opt</td>
<td>yes</td>
<td>c</td>
<td>LS</td>
<td>c 2-opt +</td>
</tr>
<tr>
<td>Local search</td>
<td>2-opt</td>
<td>no</td>
<td>c</td>
<td>LS</td>
<td>c 2-opt</td>
</tr>
<tr>
<td>Local search</td>
<td>2-opt</td>
<td>yes</td>
<td>c</td>
<td>LS</td>
<td>c 2-opt +</td>
</tr>
<tr>
<td>Iterated local search</td>
<td>jump</td>
<td>no</td>
<td>c</td>
<td>ILS</td>
<td>c jump</td>
</tr>
<tr>
<td>Iterated local search</td>
<td>jump</td>
<td>yes</td>
<td>c</td>
<td>ILS</td>
<td>c jump +</td>
</tr>
<tr>
<td>Iterated local search</td>
<td>jump</td>
<td>no</td>
<td>c</td>
<td>ILS</td>
<td>c jump</td>
</tr>
<tr>
<td>Iterated local search</td>
<td>jump</td>
<td>yes</td>
<td>c</td>
<td>ILS</td>
<td>c jump +</td>
</tr>
<tr>
<td>Iterated local search</td>
<td>swap</td>
<td>no</td>
<td>c</td>
<td>ILS</td>
<td>c swap</td>
</tr>
<tr>
<td>Iterated local search</td>
<td>swap</td>
<td>yes</td>
<td>c</td>
<td>ILS</td>
<td>c swap +</td>
</tr>
<tr>
<td>Iterated local search</td>
<td>swap</td>
<td>no</td>
<td>c</td>
<td>ILS</td>
<td>c 2-opt</td>
</tr>
<tr>
<td>Iterated local search</td>
<td>swap</td>
<td>yes</td>
<td>c</td>
<td>ILS</td>
<td>c 2-opt +</td>
</tr>
<tr>
<td>Iterated local search</td>
<td>swap</td>
<td>no</td>
<td>c</td>
<td>ILS</td>
<td>c 2-opt</td>
</tr>
<tr>
<td>Iterated local search</td>
<td>swap</td>
<td>yes</td>
<td>c</td>
<td>ILS</td>
<td>c 2-opt +</td>
</tr>
<tr>
<td>Tabu search</td>
<td>jump</td>
<td>yes</td>
<td>c</td>
<td>TS</td>
<td>c jump +</td>
</tr>
<tr>
<td>Tabu search</td>
<td>jump</td>
<td>no</td>
<td>c</td>
<td>TS</td>
<td>c jump</td>
</tr>
<tr>
<td>Tabu search</td>
<td>jump</td>
<td>yes</td>
<td>c</td>
<td>TS</td>
<td>c jump +</td>
</tr>
<tr>
<td>Tabu search</td>
<td>swap</td>
<td>yes</td>
<td>c</td>
<td>TS</td>
<td>c swap +</td>
</tr>
<tr>
<td>Tabu search</td>
<td>swap</td>
<td>no</td>
<td>c</td>
<td>TS</td>
<td>c swap</td>
</tr>
<tr>
<td>Tabu search</td>
<td>swap</td>
<td>yes</td>
<td>c</td>
<td>TS</td>
<td>c 2-opt +</td>
</tr>
<tr>
<td>Simulated annealing</td>
<td>jump</td>
<td>no</td>
<td>c</td>
<td>SA</td>
<td>c jump l.</td>
</tr>
<tr>
<td>Simulated annealing</td>
<td>jump</td>
<td>no</td>
<td>c</td>
<td>SA</td>
<td>c jump e.</td>
</tr>
<tr>
<td>Simulated annealing</td>
<td>swap</td>
<td>no</td>
<td>c</td>
<td>SA</td>
<td>c swap l.</td>
</tr>
<tr>
<td>Simulated annealing</td>
<td>swap</td>
<td>no</td>
<td>c</td>
<td>SA</td>
<td>c swap e.</td>
</tr>
<tr>
<td>Simulated annealing</td>
<td>2-opt</td>
<td>no</td>
<td>c</td>
<td>SA</td>
<td>c 2-opt l.</td>
</tr>
<tr>
<td>Simulated annealing</td>
<td>2-opt</td>
<td>no</td>
<td>c</td>
<td>SA</td>
<td>c 2-opt e.</td>
</tr>
<tr>
<td>Simulated annealing</td>
<td>3-opt</td>
<td>no</td>
<td>c</td>
<td>SA</td>
<td>c 3-opt l.</td>
</tr>
<tr>
<td>Simulated annealing</td>
<td>3-opt</td>
<td>no</td>
<td>c</td>
<td>SA</td>
<td>c 3-opt e.</td>
</tr>
</tbody>
</table>

Table 5.2: An overview of the implemented genetic algorithms.

<table>
<thead>
<tr>
<th>Class</th>
<th>Selection method</th>
<th>Crossover method</th>
<th>Mutation method</th>
<th>Label</th>
</tr>
</thead>
<tbody>
<tr>
<td>Genetic algorithm</td>
<td>roulette wheel</td>
<td>order</td>
<td>random</td>
<td>GA rw order random</td>
</tr>
<tr>
<td>Genetic algorithm</td>
<td>roulette wheel</td>
<td>order</td>
<td>swap</td>
<td>GA rw order swap</td>
</tr>
<tr>
<td>Genetic algorithm</td>
<td>roulette wheel</td>
<td>1-point</td>
<td>random</td>
<td>GA rw 1-point random</td>
</tr>
<tr>
<td>Genetic algorithm</td>
<td>roulette wheel</td>
<td>1-point</td>
<td>swap</td>
<td>GA rw 1-point swap</td>
</tr>
<tr>
<td>Genetic algorithm</td>
<td>roulette wheel</td>
<td>2-point</td>
<td>random</td>
<td>GA rw 2-point random</td>
</tr>
<tr>
<td>Genetic algorithm</td>
<td>roulette wheel</td>
<td>2-point</td>
<td>swap</td>
<td>GA rw 2-point swap</td>
</tr>
</tbody>
</table>
Chapter 6

Implementation and testing environment

This chapter describes the environment, test sets and validation set are used for this project. It substantiates why and how these test sets and the validation set are created. In Section 6.1 the benchmark environment is discussed. In Section 6.2 more information about the test and validation sets is given.

6.1 Benchmark environment

In this section we discuss the benchmark environment. We give sufficient information so that the experiments can be rerun in a similar manner. After this we discuss the use of existing toolboxes.

All benchmarks are performed in Matlab R2018b Update 2 (9.5.0.1033004). They are run on a Dell Latitude 5491 with a i7-8850H 6 core processor and 16 GB RAM. In the Matlab version code is compiled in release mode if placed in a function file. This is done for all described implementations.

Multiple packages are available nowadays that help to run all kinds of algorithms on data. For example the Matlab Optimization Toolbox Matlab Global Optimization Toolbox, IBM CPLEX, Google OR-tools, Gurobi, SCIP, GLPK, Google GLOP or Google CP-SAT. It was decided not to use these toolboxes for optimization algorithms since no toolbox with the required freedom was found and testing with different toolbox gives no realistic performance comparison.

6.2 Test and validation sets

This section describes the test sets and validation set that are created and used during this graduation project. Each test set exist out of ten problem instances, the validation set exists out of three problem instances. The test sets are used to learn the best values for the hyper-parameters. The validation set is used to show the results for the implemented algorithms on actual data. The sets are explained so that the reader has a basic understanding on the data and understands how these sets are created.

Figure 6.1 shows the number of orders that are produced on the SMD production environment per week for 25 weeks starting from week 25 of 2019 in a bar chart. In this chart it can be observed that on average approximately 100 jobs are scheduled per week.

As we described in Chapter 1, currently orders are scheduled per week. Therefore a test set of ten problem instances of 100 jobs per test case is created. All test cases have a pre-condition on each line, meaning that before the production of the first jobs scheduled a production line a changeover has to be performed. The cost and time of this changeover can be calculated similarly as the cost of all other changeovers but now based on the product produced during a static job.
In reality, the preceding orders when appending a schedule will never end at exactly the same time. Therefore for each line the preceding job is scheduled as the first job on the line. Since these jobs have a different processing time, changeovers on the different machines start at another moment in time.

Since we currently occupy seven SMD production lines, all problem instances are scheduled on seven machines.

![Figure 6.1: A bar chart of the number of order per week scheduled at the SMD PCBA manufacturing division in the upcoming weeks of 2019.](image)

As described in Chapter 1 a new planning process has developed which is expected to be introduced in Q3 of 2020. In this scheduling process the four weeks will be scheduled ahead meaning that the upcoming four weeks are present on the schedule. During this process we want to be able to reschedule all changeable jobs.

At the moment of rescheduling the shop floor planning four and a half weeks of jobs are available on the schedule. In this schedule, the upcoming two days should not be changed because warehouse may already started to prepare these jobs. Therefore four weeks of jobs can be rescheduled. Here for test set three is created which exist out of ten test cases with 400 jobs.

The last test set we use is aimed to represent a situation where we want to optimize the last two week of the schedule. This can for example come in handy when warehouse is running ahead. Here for test set two is created which contains ten test cases of 200 jobs.

Furthermore a validation set of three problem instances is created from the history of the schedule created in the Planning Viewer. For these validation cases the cost function can be evaluated for the schedule that is used for the SMD production lines. Also, for these validation cases, after running the implemented algorithms an evaluation of the cost function, can be compared with an evaluation of the cost function over the schedule that was used by production. This gives a more realistic expectation of the cost that is reduced than starting from a greedy schedule.

These validation cases contain 83, 213 and 412 jobs respectively. Their schedule that was used on the SMD production floor can be found in Figure 6.2, Figure 6.3 and Figure 6.4 respectively.

All test cases in test set one have a similar release time and deadline structure as the schedule in Figure 6.2. As indicated on average seven jobs have a deadline. This deadline is related to the maximum deadline of the test case. More precisely it is approximately half of the maximum deadline. The rest of the jobs all have the same deadline. Furthermore on average ten jobs have a release time, while the rest of the jobs have a release time of zero.
All test cases in test set three have a similar schedule to the schedule in Figure 6.4. Most jobs in this test set have a scheduling freedom of eight or nine days. Since a lot of jobs in a single test case share the same release time and deadline, an overlapping time window at the edges of the start and end of the week can be found. In this time window orders of both weeks can be scheduled. In the center of the week normally only jobs for the concerning week are scheduled.

Table 6.1 gives some basic information about all problem instances in the test and validation sets used during this graduation project.

Figure 6.2: A visualization of the actual production schedule test set 4 existing out of 83 jobs that were scheduled in 1 week.

Figure 6.3: A visualization of the actual production schedule test set four existing out of 213 jobs what were scheduled in two weeks.
Figure 6.4: A visualization of the actual production schedule test set 4 existing out of 412 jobs that were scheduled in 4 weeks.
<table>
<thead>
<tr>
<th>Set</th>
<th>Name</th>
<th># jobs</th>
<th># dist products</th>
<th>max deadline</th>
<th>dist # BOM</th>
<th>min # BOM</th>
<th>average # BOM</th>
<th>max # BOM</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>J100_01</td>
<td>100</td>
<td>90</td>
<td>289428</td>
<td>2287</td>
<td>2</td>
<td>79.26</td>
<td>165</td>
</tr>
<tr>
<td>1</td>
<td>J100_02</td>
<td>100</td>
<td>91</td>
<td>195313</td>
<td>2251</td>
<td>3</td>
<td>78.32</td>
<td>180</td>
</tr>
<tr>
<td>1</td>
<td>J100_03</td>
<td>100</td>
<td>93</td>
<td>170387</td>
<td>2809</td>
<td>3</td>
<td>80.26</td>
<td>207</td>
</tr>
<tr>
<td>1</td>
<td>J100_04</td>
<td>100</td>
<td>89</td>
<td>167304</td>
<td>2452</td>
<td>6</td>
<td>83.13</td>
<td>186</td>
</tr>
<tr>
<td>1</td>
<td>J100_05</td>
<td>100</td>
<td>83</td>
<td>253209</td>
<td>2650</td>
<td>5</td>
<td>80.04</td>
<td>207</td>
</tr>
<tr>
<td>1</td>
<td>J100_06</td>
<td>100</td>
<td>92</td>
<td>210741</td>
<td>2716</td>
<td>1</td>
<td>78.30</td>
<td>195</td>
</tr>
<tr>
<td>1</td>
<td>J100_07</td>
<td>100</td>
<td>95</td>
<td>196201</td>
<td>2655</td>
<td>1</td>
<td>78.00</td>
<td>248</td>
</tr>
<tr>
<td>1</td>
<td>J100_08</td>
<td>100</td>
<td>92</td>
<td>249838</td>
<td>3008</td>
<td>7</td>
<td>85.65</td>
<td>206</td>
</tr>
<tr>
<td>1</td>
<td>J100_09</td>
<td>100</td>
<td>90</td>
<td>207391</td>
<td>2418</td>
<td>1</td>
<td>81.3</td>
<td>310</td>
</tr>
<tr>
<td>1</td>
<td>J100_10</td>
<td>100</td>
<td>95</td>
<td>253012</td>
<td>2594</td>
<td>3</td>
<td>74.51</td>
<td>207</td>
</tr>
<tr>
<td>2</td>
<td>J200_01</td>
<td>200</td>
<td>125</td>
<td>471089</td>
<td>2555</td>
<td>1</td>
<td>84.70</td>
<td>167</td>
</tr>
<tr>
<td>2</td>
<td>J200_02</td>
<td>200</td>
<td>137</td>
<td>469069</td>
<td>2627</td>
<td>1</td>
<td>79.50</td>
<td>185</td>
</tr>
<tr>
<td>2</td>
<td>J200_03</td>
<td>200</td>
<td>145</td>
<td>468365</td>
<td>2625</td>
<td>2</td>
<td>71.28</td>
<td>175</td>
</tr>
<tr>
<td>2</td>
<td>J200_04</td>
<td>200</td>
<td>156</td>
<td>466849</td>
<td>3443</td>
<td>1</td>
<td>79.63</td>
<td>213</td>
</tr>
<tr>
<td>2</td>
<td>J200_05</td>
<td>200</td>
<td>149</td>
<td>462826</td>
<td>3061</td>
<td>6</td>
<td>83.94</td>
<td>186</td>
</tr>
<tr>
<td>2</td>
<td>J200_06</td>
<td>200</td>
<td>161</td>
<td>479315</td>
<td>3067</td>
<td>1</td>
<td>79.46</td>
<td>175</td>
</tr>
<tr>
<td>2</td>
<td>J200_07</td>
<td>200</td>
<td>166</td>
<td>529439</td>
<td>2738</td>
<td>1</td>
<td>71.12</td>
<td>180</td>
</tr>
<tr>
<td>2</td>
<td>J200_08</td>
<td>200</td>
<td>156</td>
<td>537005</td>
<td>2996</td>
<td>1</td>
<td>79.60</td>
<td>183</td>
</tr>
<tr>
<td>2</td>
<td>J200_09</td>
<td>200</td>
<td>156</td>
<td>517059</td>
<td>3050</td>
<td>1</td>
<td>76.22</td>
<td>193</td>
</tr>
<tr>
<td>2</td>
<td>J200_10</td>
<td>200</td>
<td>160</td>
<td>5051942</td>
<td>2718</td>
<td>1</td>
<td>71.48</td>
<td>175</td>
</tr>
<tr>
<td>3</td>
<td>J400_01</td>
<td>400</td>
<td>291</td>
<td>936728</td>
<td>4388</td>
<td>1</td>
<td>79.93</td>
<td>310</td>
</tr>
<tr>
<td>3</td>
<td>J400_02</td>
<td>400</td>
<td>254</td>
<td>2065385</td>
<td>3398</td>
<td>1</td>
<td>75.20</td>
<td>199</td>
</tr>
<tr>
<td>3</td>
<td>J400_03</td>
<td>400</td>
<td>266</td>
<td>9832542</td>
<td>3964</td>
<td>1</td>
<td>76.47</td>
<td>191</td>
</tr>
<tr>
<td>3</td>
<td>J400_04</td>
<td>400</td>
<td>265</td>
<td>1279264</td>
<td>4067</td>
<td>1</td>
<td>79.69</td>
<td>191</td>
</tr>
<tr>
<td>3</td>
<td>J400_05</td>
<td>400</td>
<td>265</td>
<td>1103249</td>
<td>4219</td>
<td>1</td>
<td>79.49</td>
<td>204</td>
</tr>
<tr>
<td>3</td>
<td>J400_06</td>
<td>400</td>
<td>253</td>
<td>1085492</td>
<td>3846</td>
<td>1</td>
<td>80.73</td>
<td>204</td>
</tr>
<tr>
<td>3</td>
<td>J400_07</td>
<td>400</td>
<td>264</td>
<td>994251</td>
<td>3898</td>
<td>1</td>
<td>80.45</td>
<td>191</td>
</tr>
<tr>
<td>3</td>
<td>J400_08</td>
<td>400</td>
<td>244</td>
<td>993857</td>
<td>3985</td>
<td>1</td>
<td>84.16</td>
<td>204</td>
</tr>
<tr>
<td>3</td>
<td>J400_09</td>
<td>400</td>
<td>255</td>
<td>1104976</td>
<td>4154</td>
<td>1</td>
<td>88.37</td>
<td>244</td>
</tr>
<tr>
<td>3</td>
<td>J400_10</td>
<td>400</td>
<td>258</td>
<td>1270758</td>
<td>3981</td>
<td>1</td>
<td>75.49</td>
<td>207</td>
</tr>
<tr>
<td>4</td>
<td>J083_01</td>
<td>83</td>
<td>76</td>
<td>249372</td>
<td>2085</td>
<td>1</td>
<td>71.28</td>
<td>154</td>
</tr>
<tr>
<td>4</td>
<td>J213_01</td>
<td>213</td>
<td>181</td>
<td>425390</td>
<td>3546</td>
<td>1</td>
<td>77.48</td>
<td>213</td>
</tr>
<tr>
<td>4</td>
<td>J412_01</td>
<td>412</td>
<td>279</td>
<td>479247</td>
<td>4112</td>
<td>1</td>
<td>77.05</td>
<td>213</td>
</tr>
</tbody>
</table>

Table 6.1: A table containing basic properties of the used validation and test instances.
Chapter 7

Evaluation of heuristic algorithms

This chapter describes the results that are found when experimenting with the classes of implemented heuristic and meta-heuristic algorithms described in Chapter 5. For all performed experiments, we used the test environment described in Chapter 6. The results of the performed experiments are presented and discussed.

First the local search methods are treated, in Section 7.1. Thereafter, in section Sections 7.2 to 7.4, we discuss the experiments performed using some different classes of meta-heuristic algorithms. From each class the best performing algorithm(s) is/are selected. Hereafter, in Section 7.5, experiments are discussed where in these best performing algorithms are compared. In these experiments the validation set is used instead of the test sets. Finally the performed hyper-parameter tuning is discussed.

7.1 Local search methods

In this section the experiments performed on some of the created test cases using the local search methods are presented and discussed. More information about how these local search methods are implemented can be found in Section 5.4. All experiments performed in this section do not use meta-heuristic algorithms. Therefore for all performed experiments, the cost function evaluated over the selected solutions descents monotonically over time. The experiments run until the first found local minimum is reached or the maximal time is reached.

Experiments of at most 30 minutes are performed on the first two test cases of each test set. Statistics of these experiments can be found in Tables 7.1 to 7.6 and plots of the cost descent of the selected solutions over time are given in Figures 7.1 to 7.6.

For each test case, a schedule must be created, from which the local search method starts. We decided to run all local search methods from the same start schedule so that the difference in cost descent can be visualized better. For the creation of this schedule we used the greedy schedule as it is described in Section 5.1. We discuss certain some observations that can be made based on the results of the experiments:

- **Observation:** In Tables 7.1 to 7.6 it can be seen that significantly fewer iterations are performed when experimenting with local search methods that exactly evaluate the solutions in the neighborhood of the selected solution compared to methods that use an approximate evaluation.
  - **Explanation:** This can be explained by the difference in time complexity between the evaluation of a neighbor exactly and approximately. As described in Section 3.3 the approximate evaluation can be performed in $O(1)$ time, while exact evaluation costs $O(m \cdot o_1 \cdot n)$ time.

- **Observation:** The relative difference in number of iterations between approximate and exact evaluation of solutions in the neighborhood changes significantly for the different test sets.
Explanation: This can also be explained based on the time complexity. Since the number of jobs the time of evaluating a neighbor approximately stays the same, while evaluating a neighbor exactly increases.

Remark: We expect that this is the reason why we can make the following observations for the individual test sets:

Observation: In Figure 7.1 the algorithms that use exact evaluation of the neighborhood outperform the algorithms that use approximate evaluation and in Figure 7.2 visa versa.

Explanation: For these test cases evaluating all elements in the neighborhood exactly is still tractable (this is, for later experiments, not the case). Since the objective of approximate evaluation does not have to align with the direction of the exact costs when changing the schedule, it seems logical that both approaches performs better in some cases.

Observation: When looking at the cost descent of the selected solution over time, when experimenting with test cases containing 200 jobs in Figures 7.3 and 7.4, we see approximate neighborhood evaluation performs similar to the exact neighborhood evaluation when experimenting with test case there is a big performance increase when using approximate neighborhood evaluation.

Observation: When looking at the cost descent of the selected solution over time, when experimenting with test cases containing 400 jobs in Figures 7.5 and 7.6, evaluating the neighborhood exactly is no longer tractable. When using a bounded neighborhood (meaning that jobs are not swapped far away from its origin) still some improvement steps are made, but the descent of the cost function is too slow to compete with the results of the experiments for which approximate neighborhood evaluation is used.

Observation: In the Figures 7.1 to 7.6 it can be observed that the costs of the initial solution fluctuates significantly. Test case J200-02 starts with more than twice the costs of test case J200-01, while they are composed using the same number of jobs.

Explanation: This is explained by the manner in which the test sets are created. This is done by randomly selecting a specific amount of orders from all orders that are produced by the SMD PCBA manufacturing division in history. The costs of the initial schedules that are created by the greedy algorithm therefore heavily depends on the amount of diversity in the BOM of the product types that are produced during these orders. If there is much harmony in these BOMs, the total changeover cost is low, otherwise it is high.

Observation: If we look at the trend of the descent in costs in Figures 7.1 to 7.6, approximately one fourth of the cost is reduced by the best performing local search method in all performed experiments.

Observation: It can be observed that binding the mutation method by a size when creating the neighborhood sometimes has an effect on the number of iterations per time unit.

Explanation: This is logical since this decreases the number of solutions in the neighborhood, so less solutions have to be evaluated per iteration.

Observation: For the experiments performed that use approximate neighborhood evaluation this difference seems less significant.

Explanation: This can be explained by the fast computation time of the this type of neighborhood evaluation. This is insignificant to the final evaluation in some cases.

Observation: For exact evaluation of the neighborhood and test cases of 400 jobs, running without the bound is no longer tractable.

Observation: In nearly all the presented results using local search algorithms it can be observed that algorithms using neighborhood structures build using job jump and job swap mutations performs better than the same algorithm using a neighborhood structure build with the 2-opt mutation.
• **Observation:** For nearly all performed experiments we find that after 30 minutes, the algorithms that use approximate evaluation are in a local minimum. The algorithms that use exact evaluation are not in a local minimum.

  ○ **Observation:** When looking at the test cases containing 200 jobs, we see that the best cost found of some exact methods is significantly better than approximate evaluation algorithms but descents slower.

  ○ **Observation:** For test cases containing 400 jobs the descent is so slow that we cannot see if it reaches a good schedule, but we assume that it will descend to a better minimum than the approximate algorithms.

• **Observation:** When looking at the first five minutes of the results experiments performed using a neighborhood build using jump mutation, using approximate evaluation of solutions performs best. This is if a single algorithm is required for all problem sizes.

Based on the observations we draw the following conclusions:

• A local search that uses an approximate evaluation of a neighborhood that is constructed using job swap mutations without a neighborhood constraint is recommended when optimizing a schedule in time. This can be applied if a schedule must be updated during the day when the SMD PCBA manufacturing division is working.

• For longer runs meta-heuristic algorithms can be used. Experiments performed with multiple meta-heuristic algorithms are described later in this chapter.

  ○ One of such extends the local search method by iterating it and starting from a new solution. This is called an iterated local search.

  ○ Since the local search that approximately evaluates a neighborhood structure constructed by the job jump mutation works good, it is selected for further evaluation in combination with an iterated local search.

  ○ Since the job jump and swap mutation neighborhood structures with constraint perform most consistently good, they are further evaluated in combination with an iterated local search.

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Iter. [#]</th>
<th>Impro. [#]</th>
<th>App. imp. [#]</th>
</tr>
</thead>
<tbody>
<tr>
<td>LS c jump</td>
<td>32</td>
<td>32</td>
<td></td>
</tr>
<tr>
<td>LS c jump +</td>
<td>84</td>
<td>84</td>
<td></td>
</tr>
<tr>
<td>LS c jump +</td>
<td>743</td>
<td>55</td>
<td>213</td>
</tr>
<tr>
<td>LS c swap</td>
<td>32</td>
<td>32</td>
<td></td>
</tr>
<tr>
<td>LS c swap +</td>
<td>94</td>
<td>74</td>
<td></td>
</tr>
<tr>
<td>LS c swap +</td>
<td>639</td>
<td>79</td>
<td>225</td>
</tr>
<tr>
<td>LS c swap +</td>
<td>422</td>
<td>60</td>
<td>153</td>
</tr>
<tr>
<td>LS c 2-opt</td>
<td>37</td>
<td>37</td>
<td></td>
</tr>
<tr>
<td>LS c 2-opt +</td>
<td>101</td>
<td>101</td>
<td></td>
</tr>
<tr>
<td>LS c 2-opt +</td>
<td>1029</td>
<td>43</td>
<td>422</td>
</tr>
<tr>
<td>LS c 2-opt +</td>
<td>668</td>
<td>39</td>
<td>306</td>
</tr>
</tbody>
</table>

Table 7.1: Statistics of the experiments performed using the different local search methods and test case J100-01. The cost descent measured during these experiments can be found in Figure 7.1.

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Iter. [#]</th>
<th>Impro. [#]</th>
<th>App. imp. [#]</th>
</tr>
</thead>
<tbody>
<tr>
<td>LS c jump</td>
<td>35</td>
<td>35</td>
<td></td>
</tr>
<tr>
<td>LS c jump +</td>
<td>89</td>
<td>89</td>
<td></td>
</tr>
<tr>
<td>LS c jump +</td>
<td>722</td>
<td>74</td>
<td>185</td>
</tr>
<tr>
<td>LS c jump +</td>
<td>599</td>
<td>69</td>
<td>192</td>
</tr>
<tr>
<td>LS c swap</td>
<td>35</td>
<td>33</td>
<td></td>
</tr>
<tr>
<td>LS c swap +</td>
<td>93</td>
<td>59</td>
<td></td>
</tr>
<tr>
<td>LS c swap +</td>
<td>579</td>
<td>110</td>
<td>208</td>
</tr>
<tr>
<td>LS c swap +</td>
<td>639</td>
<td>76</td>
<td>150</td>
</tr>
<tr>
<td>LS c 2-opt</td>
<td>40</td>
<td>40</td>
<td></td>
</tr>
<tr>
<td>LS c 2-opt +</td>
<td>98</td>
<td>98</td>
<td></td>
</tr>
<tr>
<td>LS c 2-opt +</td>
<td>2062</td>
<td>61</td>
<td>848</td>
</tr>
<tr>
<td>LS c 2-opt +</td>
<td>954</td>
<td>68</td>
<td>421</td>
</tr>
</tbody>
</table>

Table 7.2: Statistics of the experiments performed using the different local search methods and test case J100-02. The cost descent measured during these experiments can be found in Figure 7.2.
Figure 7.1: The descent of costs measured during experiments performed using different local search methods and test case J100-01.

Figure 7.2: The descent of costs measured during experiments performed using different local search methods and test case J100-02.

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Iter. [#]</th>
<th>Impro. [#]</th>
<th>App. imp. [#]</th>
</tr>
</thead>
<tbody>
<tr>
<td>LS ⋅ jump</td>
<td>7</td>
<td>7</td>
<td></td>
</tr>
<tr>
<td>LS ⋅ jump +</td>
<td>38</td>
<td>38</td>
<td></td>
</tr>
<tr>
<td>LS ⋅ jump</td>
<td>1486</td>
<td>103</td>
<td>421</td>
</tr>
<tr>
<td>LS ⋅ jump +</td>
<td>1474</td>
<td>110</td>
<td>344</td>
</tr>
<tr>
<td>LS ⋅ swap</td>
<td>7</td>
<td>7</td>
<td></td>
</tr>
<tr>
<td>LS ⋅ swap +</td>
<td>38</td>
<td>37</td>
<td></td>
</tr>
<tr>
<td>LS ⋅ swap</td>
<td>1419</td>
<td>108</td>
<td>486</td>
</tr>
<tr>
<td>LS ⋅ swap +</td>
<td>1166</td>
<td>89</td>
<td>344</td>
</tr>
<tr>
<td>LS ⋅ 2-opt</td>
<td>8</td>
<td>8</td>
<td></td>
</tr>
<tr>
<td>LS ⋅ 2-opt +</td>
<td>42</td>
<td>41</td>
<td></td>
</tr>
<tr>
<td>LS ⋅ 2-opt</td>
<td>4376</td>
<td>275</td>
<td>1832</td>
</tr>
<tr>
<td>LS ⋅ 2-opt +</td>
<td>677</td>
<td>15</td>
<td>249</td>
</tr>
</tbody>
</table>

Table 7.3: Statistics of the experiments performed using the different local search methods and test case J200-01. The cost descent measured during these experiments can be found in Figure 7.3.

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Iter. [#]</th>
<th>Impro. [#]</th>
<th>App. imp. [#]</th>
</tr>
</thead>
<tbody>
<tr>
<td>LS ⋅ jump</td>
<td>11</td>
<td>11</td>
<td></td>
</tr>
<tr>
<td>LS ⋅ jump +</td>
<td>53</td>
<td>53</td>
<td></td>
</tr>
<tr>
<td>LS ⋅ jump</td>
<td>1123</td>
<td>65</td>
<td>359</td>
</tr>
<tr>
<td>LS ⋅ jump +</td>
<td>893</td>
<td>52</td>
<td>212</td>
</tr>
<tr>
<td>LS ⋅ swap</td>
<td>11</td>
<td>11</td>
<td></td>
</tr>
<tr>
<td>LS ⋅ swap +</td>
<td>54</td>
<td>54</td>
<td></td>
</tr>
<tr>
<td>LS ⋅ swap</td>
<td>909</td>
<td>45</td>
<td>423</td>
</tr>
<tr>
<td>LS ⋅ swap +</td>
<td>1124</td>
<td>59</td>
<td>257</td>
</tr>
<tr>
<td>LS ⋅ 2-opt</td>
<td>13</td>
<td>13</td>
<td></td>
</tr>
<tr>
<td>LS ⋅ 2-opt +</td>
<td>59</td>
<td>59</td>
<td></td>
</tr>
<tr>
<td>LS ⋅ 2-opt</td>
<td>2884</td>
<td>76</td>
<td>1580</td>
</tr>
<tr>
<td>LS ⋅ 2-opt +</td>
<td>5505</td>
<td>77</td>
<td>2392</td>
</tr>
</tbody>
</table>

Table 7.4: Statistics of the experiments performed using the different local search methods and test case J200-02. The cost descent measured during these experiments can be found in Figure 7.4.
Figure 7.3: The descent of costs measured during experiments performed using different local search methods and test case J200-01.

Figure 7.4: The descent of costs measured during experiments performed using different local search methods and test case J200-02.

Table 7.5: Statistics of the experiments performed using the different local search methods and test case J400-01. The cost descent measured during these experiments can be found in Figure 7.5.

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Iter. [#]</th>
<th>Impr. [#]</th>
<th>App. imp. [#]</th>
</tr>
</thead>
<tbody>
<tr>
<td>LS + jump</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>LS + jump +</td>
<td>12</td>
<td>12</td>
<td></td>
</tr>
<tr>
<td>LS + jump</td>
<td>2433</td>
<td>134</td>
<td>1278</td>
</tr>
<tr>
<td>LS + swap</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>LS + swap +</td>
<td>11</td>
<td>10</td>
<td></td>
</tr>
<tr>
<td>LS + swap</td>
<td>2715</td>
<td>190</td>
<td>1832</td>
</tr>
<tr>
<td>LS + swap +</td>
<td>4002</td>
<td>255</td>
<td>1233</td>
</tr>
<tr>
<td>LS + 2-opt</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>LS + 2-opt +</td>
<td>13</td>
<td>13</td>
<td></td>
</tr>
<tr>
<td>LS + 2-opt</td>
<td>1363</td>
<td>682</td>
<td>690</td>
</tr>
<tr>
<td>LS + 2-opt +</td>
<td>1070</td>
<td>32</td>
<td>398</td>
</tr>
</tbody>
</table>

Table 7.6: Statistics of the experiments performed using the different local search methods and test case J400-02. The cost descent measured during these experiments can be found in Figure 7.6.

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Iter. [#]</th>
<th>Impr. [#]</th>
<th>App. imp. [#]</th>
</tr>
</thead>
<tbody>
<tr>
<td>LS + jump</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>LS + jump +</td>
<td>15</td>
<td>15</td>
<td></td>
</tr>
<tr>
<td>LS + jump</td>
<td>3410</td>
<td>227</td>
<td>1449</td>
</tr>
<tr>
<td>LS + jump +</td>
<td>5885</td>
<td>254</td>
<td>1473</td>
</tr>
<tr>
<td>LS + swap</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>LS + swap +</td>
<td>15</td>
<td>15</td>
<td></td>
</tr>
<tr>
<td>LS + swap</td>
<td>4553</td>
<td>326</td>
<td>2353</td>
</tr>
<tr>
<td>LS + swap +</td>
<td>3762</td>
<td>192</td>
<td>1156</td>
</tr>
<tr>
<td>LS + 2-opt</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>LS + 2-opt +</td>
<td>18</td>
<td>18</td>
<td></td>
</tr>
<tr>
<td>LS + 2-opt</td>
<td>2146</td>
<td>50</td>
<td>712</td>
</tr>
<tr>
<td>LS + 2-opt +</td>
<td>2258</td>
<td>29</td>
<td>412</td>
</tr>
</tbody>
</table>
7.2 Tabu search

In this section the experiments performed on some of the created test cases using tabu search algorithms are given and discussed. More information about how these tabu search algorithms are implemented can be found in Section 5.5.

Experiments of 60 minutes are performed on the first two test cases of each test set. Statistics about these experiments can be found in Table 7.7. Plots of the cost of the selected solution over time can be found in Figures 7.7, 7.9, 7.11, 7.13, 7.15 and 7.17. Plots of the cost descent of the incumbent solutions measured over time during the same experiments can be found in Figures 7.8, 7.10, 7.12, 7.14, 7.16 and 7.18.

Similar as for local search methods, for each test case, an initial schedule must be created. The tabu search algorithms starts from this schedule. During these experiment, this schedule is created in the same manner as for local search methods, namely using the described greedy algorithm. We now give the observations that are made on the results of the experiments performed using the class of tabu search algorithms:

- **Remark:** Experiments using an exact evaluation of the solutions in the neighborhood rather than approximate method have been performed. For these experiments no plots are presented because there is not so much to show. In these experiments, the costs of all solutions in the neighborhood is calculated exactly. The neighborhood is constructed using a job jump or swap mutation. These mutations are bound by a short distance in time (four jobs).
- **Observation:** When running these experiments, not a single neighborhood can be evaluated within half an hour.
- **Explanation:** The lack in performance can be explained by the time complexity of evaluation the cost of a solution exactly.
- **Observation:** In Table 7.7 we can find the number of iterations performed per experiment. We can observe that when introducing a neighborhood size this number does not always decrease significantly.
  - **Remark:** Normally decreasing the neighborhood size increases this number significantly. This is because it significantly decreases the number of solutions in the neighborhood so less solutions have to be evaluated.
  - **Explanation:** This can be explained by the insignificance in computation time of the neighborhood evaluation part of an iteration. Since nearly all computation is spent on
exact evaluation of the selected neighbor, the number of solutions that must be evaluated
approximately does not matter so much.

- **Remark:** Experiments have been performed using a tabu search algorithm that uses an approxi-
mation of the cost by only considering the changeover difference.
  - **Observation:** When solely using the difference in changeover as an approximation of the
costs, the exact costs kept increasing over time.
    - **Explanation:** After evaluating the produced schedules over time, it was found that
      more and more jobs where placed on a position where they are late. These jobs will
      never be moved by because they will never have best approximate improvement. This
      is because there was no intrinsic in place that moves them within their bounds again.
  - **Remark:** This was expected to be a sufficient approximation. If it is a good enough
    approximation, it will find a good direction for the change in costs.
  - **Remark:** From these experiments the need of including a fast tardiness approximation
    of the tardiness delta was found. Using the new approximation cost function neighbor
    performs significantly better.

- **Observation:** What can be observed in Figures 7.7 and 7.11 is that during these experiments,
  for multiple algorithms, multiple solutions are reached that have approximately the same cost.
  - **Explanation:** When looking at these figures it is expected that these solutions have the
    same wrong move in common that increase the cost function to a certain level. Since when
    something else in this schedule changes, the resulting schedule after doing this bad move
    again will be different and this is again evaluated.
  - **Remark:** This is a consequence of having solutions in the tabu list rather than moves.
    Based on this observation, more experiments may be beneficial with an divergence of
    the tabu algorithm. This algorithm should place moves in the tabu list rather than whole
    schedules.

- **Observation:** What can be observed in the statistics is that for all performed experiments with
  the tabu search algorithms, the number of approximate improvements is the same as the number
  of iterations.
  - **Explanation:** This means that in no case a point is reached where no approximate
    improving move exist.
  - **Remark:** Reaching such a point is supposed to happen more often when applying a tabu
    search algorithm. This should especially happen when a tabu search is combined with a
    limited neighborhood structure. This is expected to be a consequence of the previous point.

- **Observation:** In Figures 7.7 to 7.18 we can see that for all performed experiments the tabu
  search algorithm, using an non constraint neighborhood structure created by performing job
  mutations decreases the most costs.

Conclusion: since experiments performed with a job mutation neighborhood structure that is unbound
in size have the best results this algorithm is further evaluated.
Table 7.7: Statistics of the experiments performed using tabu search algorithms. The change in costs over time for the selected and incumbent solution measured during these experiments can be found in Figures 7.7 to 7.18.

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Test case</th>
<th>Iter. [#]</th>
<th>Impr. [#]</th>
<th>App. impr. [#]</th>
<th>Bad move [#]</th>
<th>Tabu list [#]</th>
</tr>
</thead>
<tbody>
<tr>
<td>TS (\tilde{c}) jump</td>
<td>J100-01</td>
<td>3391</td>
<td>29</td>
<td>3391</td>
<td>1465</td>
<td>1955</td>
</tr>
<tr>
<td>TS (\tilde{c}) jump</td>
<td>J100-02</td>
<td>8202</td>
<td>63</td>
<td>8202</td>
<td>717</td>
<td>7915</td>
</tr>
<tr>
<td>TS (\tilde{c}) jump</td>
<td>J100-01</td>
<td>3016</td>
<td>29</td>
<td>3016</td>
<td>1657</td>
<td>1673</td>
</tr>
<tr>
<td>TS (\tilde{c}) jump</td>
<td>J100-02</td>
<td>4011</td>
<td>58</td>
<td>4011</td>
<td>2101</td>
<td>2312</td>
</tr>
<tr>
<td>TS (\tilde{c}) swap</td>
<td>J100-01</td>
<td>2963</td>
<td>15</td>
<td>2963</td>
<td>1513</td>
<td>1513</td>
</tr>
<tr>
<td>TS (\tilde{c}) swap</td>
<td>J100-02</td>
<td>3638</td>
<td>1</td>
<td>3638</td>
<td>1817</td>
<td>1817</td>
</tr>
<tr>
<td>TS (\tilde{c}) swap</td>
<td>J100-01</td>
<td>1569</td>
<td>23</td>
<td>1569</td>
<td>970</td>
<td>970</td>
</tr>
<tr>
<td>TS (\tilde{c}) swap</td>
<td>J100-02</td>
<td>3573</td>
<td>7</td>
<td>3573</td>
<td>1735</td>
<td>1735</td>
</tr>
<tr>
<td>TS (\tilde{c}) jump</td>
<td>J200-01</td>
<td>1846</td>
<td>37</td>
<td>1846</td>
<td>1104</td>
<td>1104</td>
</tr>
<tr>
<td>TS (\tilde{c}) jump</td>
<td>J200-02</td>
<td>1891</td>
<td>4</td>
<td>1891</td>
<td>984</td>
<td>984</td>
</tr>
<tr>
<td>TS (\tilde{c}) jump</td>
<td>J200-01</td>
<td>1940</td>
<td>21</td>
<td>1940</td>
<td>1002</td>
<td>1002</td>
</tr>
<tr>
<td>TS (\tilde{c}) jump</td>
<td>J200-02</td>
<td>2060</td>
<td>3</td>
<td>2060</td>
<td>1116</td>
<td>1116</td>
</tr>
<tr>
<td>TS (\tilde{c}) swap</td>
<td>J200-01</td>
<td>1904</td>
<td>22</td>
<td>1904</td>
<td>1011</td>
<td>1011</td>
</tr>
<tr>
<td>TS (\tilde{c}) swap</td>
<td>J200-02</td>
<td>2050</td>
<td>1</td>
<td>2050</td>
<td>1066</td>
<td>1066</td>
</tr>
<tr>
<td>TS (\tilde{c}) swap</td>
<td>J200-01</td>
<td>1855</td>
<td>14</td>
<td>1855</td>
<td>1002</td>
<td>1002</td>
</tr>
<tr>
<td>TS (\tilde{c}) swap</td>
<td>J200-02</td>
<td>1963</td>
<td>0</td>
<td>1963</td>
<td>1054</td>
<td>1054</td>
</tr>
<tr>
<td>TS (\tilde{c}) jump</td>
<td>J400-01</td>
<td>2094</td>
<td>0</td>
<td>2094</td>
<td>288</td>
<td>1843</td>
</tr>
<tr>
<td>TS (\tilde{c}) jump</td>
<td>J400-02</td>
<td>1365</td>
<td>36</td>
<td>1365</td>
<td>495</td>
<td>990</td>
</tr>
<tr>
<td>TS (\tilde{c}) jump</td>
<td>J400-01</td>
<td>2810</td>
<td>0</td>
<td>2810</td>
<td>204</td>
<td>2623</td>
</tr>
<tr>
<td>TS (\tilde{c}) jump</td>
<td>J400-02</td>
<td>989</td>
<td>1</td>
<td>989</td>
<td>561</td>
<td>564</td>
</tr>
<tr>
<td>TS (\tilde{c}) swap</td>
<td>J400-01</td>
<td>2074</td>
<td>0</td>
<td>2074</td>
<td>397</td>
<td>1750</td>
</tr>
<tr>
<td>TS (\tilde{c}) swap</td>
<td>J400-02</td>
<td>1015</td>
<td>1</td>
<td>1015</td>
<td>524</td>
<td>536</td>
</tr>
<tr>
<td>TS (\tilde{c}) swap</td>
<td>J400-01</td>
<td>2817</td>
<td>0</td>
<td>2817</td>
<td>181</td>
<td>2656</td>
</tr>
<tr>
<td>TS (\tilde{c}) swap</td>
<td>J400-02</td>
<td>998</td>
<td>0</td>
<td>998</td>
<td>551</td>
<td>551</td>
</tr>
</tbody>
</table>

Figure 7.7: The change in costs of the exact evaluated solutions over time measured during experiments performed using tabu search algorithms and test case J100-01.

Figure 7.8: The descent of the costs of the incumbent solution measured during experiments performed using tabu search algorithms and test case J100-01.
Figure 7.9: The change in costs of the exact evaluated solutions over time measured during experiments performed using tabu search algorithms and test case J100-02.

Figure 7.10: The descent of the costs of the incumbent solution measured during experiments performed using tabu search algorithms and test case J100-02.

Figure 7.11: The change in costs of the exact evaluated solutions over time measured during experiments performed using tabu search algorithms and test case J200-01.

Figure 7.12: The descent of the costs of the incumbent solution measured during experiments performed using tabu search algorithms and test case J200-01.
Figure 7.13: The change in costs of the exact evaluated solutions over time measured during experiments performed using tabu search algorithms and test case J200-02.

Figure 7.14: The descent of the costs of the incumbent solution measured during experiments performed using tabu search algorithms and test case J200-02.

Figure 7.15: The change in costs of the exact evaluated solutions over time measured during experiments performed using tabu search algorithms and test case J400-01.

Figure 7.16: The descent of the costs of the incumbent solution measured during experiments performed using tabu search algorithms and test case J400-01.
7.3 Simulated annealing

In this section some experiments performed on problem instances in the test sets using simulated annealing are described. We looked at the performance when combining neighborhood structures and cooling schemes as described in Section 5.6.

Experiments of 60 minutes are performed on the first two test cases of each test set. Statistics about these experiments can be found in Tables 7.8 to 7.13. Plots of the cost of the selected solution over time can be found in Figures 7.19 to 7.24.

Similar as for local search and tabu search algorithms, an initial schedule must be created. For the experiments performed with simulated annealing we used a random and greedy schedule. The presented figures and tables present measurements performed starting from greedy schedules. We now give the observations that are made on the results of the experiments performed using the class of simulated annealing algorithms:

- **Observation:** When experimenting with simulated annealing with a first schedule that is created using the random method, it can be observed that the cost of this starting point is so high that the cost will it decrease enough in to find better values than a greedy created schedule.

- **Observation:** As can be seen in Figures 7.19 and 7.23 the descent of the trend of the costs when experimenting with simulated annealing is way steeper for test cases of 100 jobs than for test cases of 400 jobs.

- **Observation:** The most important observation that can be made is that running the simulated annealing algorithm using a 2-opt or 3-opt neighborhood structure performs significantly worse than when using a job jump or job swap neighborhood structure.
  - **Observation:** In the statistics it can be found that significantly less improvement steps and bad moves are made, while the number of iterations are the same.
  - **Remark:** From this we conclude that k-opt neighborhood structures are less suitable when applying simulated annealing to the concerning scheduling problem.

- **Observation:** When looking at Figures 7.19 to 7.24 it can be observed that the slope for different problem sizes varies.
  - **Observation:** In Figure 7.19 it can be observed that the fitness of the incumbent solution decreases significantly for most used neighborhood structures in the first 400 seconds for problem instances with 100 jobs.
Observation: In Figure 7.23 it can be observed that the fitness of the incumbent solution decreases similarly over time.

Remark: When looking at these figures, for problem instances of 100 jobs, the balance between exploitation and exploration seems to be too much in the direction of exploitation while for problem instances of 400 jobs the balance seems to be too much in the direction of exploration.

Observation: It can be observed in Figures 7.19 to 7.24 that the runs performed using exponential cooling scheme perform similar or better than experiments performed on the same test case with the same neighborhood structure and a linear cooling scheme.

Conclusion: we use simulated annealing with a job jump and job swap mutation using an exponential cooling scheme for further experiments.

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Iter. [#]</th>
<th>Impro. [#]</th>
<th>Bad move [#]</th>
<th>Algorithm</th>
<th>Iter. [#]</th>
<th>Impro. [#]</th>
<th>Bad move [#]</th>
</tr>
</thead>
<tbody>
<tr>
<td>SA c jump l.</td>
<td>5840</td>
<td>1105</td>
<td>1267</td>
<td>SA c jump l.</td>
<td>5836</td>
<td>833</td>
<td>1055</td>
</tr>
<tr>
<td>SA c jump e.</td>
<td>6416</td>
<td>165</td>
<td>38</td>
<td>SA c jump e.</td>
<td>6900</td>
<td>185</td>
<td>36</td>
</tr>
<tr>
<td>SA c swap l.</td>
<td>5611</td>
<td>1486</td>
<td>1131</td>
<td>SA c swap l.</td>
<td>5483</td>
<td>1215</td>
<td>933</td>
</tr>
<tr>
<td>SA c swap e.</td>
<td>6400</td>
<td>264</td>
<td>32</td>
<td>SA c swap e.</td>
<td>6856</td>
<td>231</td>
<td>41</td>
</tr>
<tr>
<td>SA c 2-opt l.</td>
<td>6283</td>
<td>282</td>
<td>212</td>
<td>SA c 2-opt l.</td>
<td>6542</td>
<td>440</td>
<td>300</td>
</tr>
<tr>
<td>SA c 2-opt e.</td>
<td>6445</td>
<td>88</td>
<td>5</td>
<td>SA c 2-opt e.</td>
<td>6899</td>
<td>131</td>
<td>11</td>
</tr>
<tr>
<td>SA c 3-opt l.</td>
<td>6410</td>
<td>59</td>
<td>35</td>
<td>SA c 3-opt l.</td>
<td>6807</td>
<td>151</td>
<td>105</td>
</tr>
<tr>
<td>SA c 3-opt e.</td>
<td>6432</td>
<td>56</td>
<td>2</td>
<td>SA c 3-opt e.</td>
<td>6908</td>
<td>58</td>
<td>2</td>
</tr>
</tbody>
</table>

Table 7.8: Statistics of the experiments performed using simulated annealing and test case J100-01. The change in costs over time measured during these experiments can be found in Figure 7.19.

Table 7.9: Statistics of the experiments performed using simulated annealing and test case J100-02. The change in costs over time measured during these experiments can be found in Figure 7.20.

Figure 7.19: The change in costs of the evaluated solutions over time measured during experiments performed using simulated annealing and test case J100-01.

Figure 7.20: The change in costs of the evaluated solutions over time measured during experiments performed using simulated annealing and test case J100-02.
Table 7.10: Statistics of the experiments performed using simulated annealing and test case J200-01. The change in costs over time measured during these experiments can be found in Figure 7.21.

Table 7.11: Statistics of the experiments performed using simulated annealing and test case J200-02. The change in costs over time measured during these experiments can be found in Figure 7.22.

Figure 7.21: The change in costs of the evaluated solutions over time measured during experiments performed using simulated annealing and test case J200-01.

Figure 7.22: The change in costs of the evaluated solutions over time measured during experiments performed using simulated annealing and test case J200-02.

Table 7.12: Statistics of the experiments performed using simulated annealing and test case J400-01. The change in costs over time measured during these experiments can be found in Figure 7.24.

Table 7.13: Statistics of the experiments performed using simulated annealing and test case J400-02. The change in costs over time measured during these experiments can be found in Figure 7.24.
7.4 Genetic algorithm

In this section the experiments performed of the test sets using genetic algorithms are discussed. More information about how these genetic algorithms are implemented can be found in Section 5.7. All experiments are performed with a population size of 32, 16 crossovers and eight permutations. Furthermore all algorithms start with the same initial population which is created as describe in Section 5.7.1. In Tables 7.14 to 7.19 some statistics can be found about how these experiments performed.

- **Observation**: The most important observation made on the statistics in Tables 7.14 to 7.19 is that there are not enough improvements made when experimenting for half an hour per algorithm.

  - **Remark**: Since one of the mutation methods implemented adds random schedules to the population and most of the found improvements over all performed experiments are found on the experiments that use this mutation methods, it is highly likely that all improvements made are caused by accidental good randoms.

- **Observation**: Furthermore it can be observed in Tables 7.14 to 7.19 that the mutation stage must take a significant amount of time for the experiments performed with random swap mutations. This is because the experiments performed with swap mutation perform significantly more iterations per time unit.

It can be concluded that the all implemented genetic algorithms perform to bad and therefor the other meta-heuristics are preferred over this meta-heuristic. We continue to use a single genetic algorithm so that it is still represented in our final experiments. Since these experiments are longer this may yield some unexpected results. The selected genetic algorithm uses 1-point crossover and random as mutation method.
Table 7.14: Statistics of the experiments performed using genetic algorithms and test case J100-01. The change in costs over time measured during these experiments can be found in Figure 7.25.

Table 7.15: Statistics of the experiments performed using genetic algorithms and test case J100-02. The change in costs over time measured during these experiments can be found in Figure 7.26.

Figure 7.25: The descent of the cost function for the different implementations of genetic algorithms for test case J100-01.

Figure 7.26: The descent of the cost function for the different implementations of genetic algorithms for test case J100-02.

Table 7.16: Statistics of the experiments performed using genetic algorithms and test case J200-01. The change in costs over time measured during these experiments can be found in Figure 7.27.

Table 7.17: Statistics of the experiments performed using genetic algorithms and test case J200-02. The change in costs over time measured during these experiments can be found in Figure 7.28.
Figure 7.27: The descent of the cost function for the different implementations of genetic algorithms for test case J200-01.

Figure 7.28: The descent of the cost function for the different implementations of genetic algorithms for test case J200-02.

Table 7.18: Statistics of the experiments performed using genetic algorithms and test case J400-01. The change in costs over time measured during these experiments can be found in Figure 7.29.

Table 7.19: Statistics of the experiments performed using genetic algorithms and test case J400-02. The change in costs over time measured during these experiments can be found in Figure 7.30.

Figure 7.29: The descent of the cost function for the different implementations of genetic algorithms for test case J400-01.

Figure 7.30: The descent of the cost function for the different implementations of genetic algorithms for test case J400-02.
7.5 Final results

In this section we describe the experiments that are performed using the created validation set. For short runs of a few minutes, meta-heuristic algorithms are not practical. Since we found in Appendix D.2 which local search method performs best, we do not further investigate this. It remains to decide which (meta-)heuristic algorithm performs best when a longer optimization time is available.

In this section, we further investigate this situation.

We perform experiments of six hours using the selected (meta-)heuristic algorithms. This is because six hours will always be available when last changes are made in the evening and the run must be finished before the morning shift starts.

First we summarize the heuristic algorithms that are selected for these experiments. These algorithms are selected because they performed best in the earlier described experiments or are expected to perform well on the created validation cases. The selected heuristic algorithms are:

- ILS $\tilde{c}$ jump
- ILS $c$ jump +
- ILS $c$ swap +
- TS $\tilde{c}$ jump
- SA $c$ jump e.
- SA $c$ swap e.
- GA rw 1-point random

Note that we did not include the cost descent of the incumbent solutions measured when running experiments with genetic algorithms on test case J213 and J412. This is because the cost of this solution is so high that this makes the plots unreadable.

Table 7.20 gives some statistics about the performed experiments. Figures 7.31 to 7.33 show the cost descent of the incumbent solution measured during these experiments. Based on these results we can make the following observations:

- **Observation**: The cost of the incumbent solution when experimenting with the selected genetic algorithm does not seem to improve at all. This may be explainable by that crossover introduces a lot of non feasible solutions.

- **Observation**: The selected tabu search algorithms perform best for the validation case with the largest size. A very good solution is found within one hour. After this the algorithm is expected to evaluate solutions around a local minimum for the rest of the experiment.

- **Observation**: The performance of the tabu algorithm is very dependent on how long it is stuck in a local minimum. In the performed experiments we see that it gets out of a local minimum and reaches a better area approximately two times in six hours.

- **Observation**: The iterated local search algorithms that use exact cost evaluation do not find a single local minimum for test cases of mid or high size.

- **Observation**: It can be observed that a simulated annealing algorithm that uses job swaps to create its neighbors performs better than when a neighborhood constructed by performing a job jump is used. This is strange since less schedules can be reached. Therefore this is expected to change when performing longer experiments.

Conclusion: it is recommended to implement a version of simulated annealing that moves to neighbors by performing job swap mutations and uses a exponential cooling scheme for optimization runs of six hours. Alternatively a tabu search algorithms that uses a neighborhood structure constructed by job jumps can be used. The selected version of the class of simulated annealing algorithms is preferred over the selected tabu search algorithm because it has a more reliable descent.
<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Test case</th>
<th>Iter. [#]</th>
<th>Impro. [#]</th>
<th>App. impr. [#]</th>
<th>Pert. [#]</th>
<th>Bad move [#]</th>
<th>Tabu [#]</th>
</tr>
</thead>
<tbody>
<tr>
<td>ILS $\tilde{c}$ jump</td>
<td>J083-01</td>
<td>161912</td>
<td>22337</td>
<td>34070</td>
<td>300</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ILS $c$ jump +</td>
<td>J083-01</td>
<td>2107</td>
<td>2107</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ILS $c$ swap +</td>
<td>J083-01</td>
<td>1749</td>
<td>350</td>
<td>2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TS $\tilde{c}$ jump</td>
<td>J083-01</td>
<td>16629</td>
<td>70</td>
<td>16447</td>
<td>14344</td>
<td>14363</td>
<td></td>
</tr>
<tr>
<td>SA $c$ jump e.</td>
<td>J083-01</td>
<td>52093</td>
<td>316</td>
<td>123</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SA $c$ swap e.</td>
<td>J083-01</td>
<td>51861</td>
<td>361</td>
<td>91</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GA rw 1-point ran.</td>
<td>J083-01</td>
<td>2503</td>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ILS $\tilde{c}$ jump</td>
<td>J213-01</td>
<td>100347</td>
<td>7902</td>
<td>40073</td>
<td>52</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ILS $c$ jump +</td>
<td>J213-01</td>
<td>338</td>
<td>338</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ILS $c$ swap +</td>
<td>J213-01</td>
<td>341</td>
<td>152</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TS $\tilde{c}$ jump</td>
<td>J213-01</td>
<td>10849</td>
<td>22</td>
<td>10849</td>
<td>2829</td>
<td>8629</td>
<td></td>
</tr>
<tr>
<td>SA $c$ jump e.</td>
<td>J213-01</td>
<td>15906</td>
<td>391</td>
<td>69</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SA $c$ swap e.</td>
<td>J213-01</td>
<td>15645</td>
<td>342</td>
<td>36</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GA rw 1-point ran.</td>
<td>J213-01</td>
<td>587</td>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ILS $\tilde{c}$ jump</td>
<td>J412-01</td>
<td>41802</td>
<td>3170</td>
<td>41950</td>
<td>12</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ILS $c$ jump +</td>
<td>J412-01</td>
<td>136</td>
<td>119</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ILS $c$ swap +</td>
<td>J412-01</td>
<td>141</td>
<td>124</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TS $\tilde{c}$ jump</td>
<td>J412-01</td>
<td>4947</td>
<td>26</td>
<td>4947</td>
<td>2475</td>
<td>3186</td>
<td></td>
</tr>
<tr>
<td>SA $c$ jump e.</td>
<td>J412-01</td>
<td>7076</td>
<td>269</td>
<td>54</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SA $c$ swap e.</td>
<td>J412-01</td>
<td>7073</td>
<td>301</td>
<td>21</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GA rw 1-point ran.</td>
<td>J412-01</td>
<td>180</td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 7.20: Statistics of the experiments performed using heuristic algorithms on the validation set. The cost descent of the incumbent solution measured during these experiments can be found in Figures 7.31 to 7.33.

Figure 7.31: The cost descent of the incumbent solutions over time for longer experiments with test case J083.
Figure 7.32: The cost descent of the incumbent solutions over time for experiments with test case J213.

Figure 7.33: The cost descent of the incumbent solutions over time for experiments with test case J412.
7.6 Hyper-parameter tuning

There are multiple hyper-parameters used by the described models. The combination of the used hyper-parameters used by the different models can be optimized in different manners. Picking these parameters poorly can demolish the gain in cost function achieved by these algorithms. First a description of all hyper-parameters used by the described algorithms is given:

- **Neighborhood size:** the neighborhood size indicates how much neighbors are evaluated. A smaller neighborhood size decreases the amount of calculation time per iteration. However it reduces the quality of the best neighbor and may cause an earlier local minimum. Finding the best balanced is crucial when using a neighborhood.

- **Tabu list length:** a small tabu list may cause only looking at a small subset of the cost with the highest cost. A to long tabu list may cause ignorance of the most expensive jobs for too long. Therefore also for tabu algorithms finding the optimum tabu list length is crucial.

- **Cooling scheme:** all implemented simulated annealing algorithms use a cooling scheme. This cooling scheme uses a hyper-parameter that is used to calculate how big the chance of performing a bad move must be.

- **Population size:** as described in Chapter 5 the population size the number of chromosomes that are maintained by a genetic algorithm. Therefore this parameter only influences genetic algorithms. The disadvantage of a big population size is that it significantly increases the calculation time per iteration. The advantage is that more solutions are evaluated, crossed and mutated.

- **Selection size:** similar to the population size, the selection size is only a parameter for genetic algorithms. When selecting more items out of the population during the run of a GA, it has a higher chance that good solutions are preserved. The disadvantage is that there is less room for worse solutions.

All described hyper-parameters are expected to be mutually dependent, meaning that if one is changed, the other can be re-optimized. Therefore the results are expected when all hyper-parameters are tuned at the same time and for each of the three test sets.

Hyper-parameter tuning is performed using grid search. For this a very short grid is used so that it can be performed in reasonable time. An example of the results of a hyper-parameter tuning run performed for a iterative local search that uses approximate evaluation of a job jump neighborhood structure can be found in Section 7.6.
Chapter 8

Conclusions and recommendations

Many optimization problems for specific practical scheduling problems result in a huge design space. Drawing conclusions about the quality of the available algorithms can only be done when these are thoroughly evaluated and when parameters like neighborhood and population size are correctly picked and substantiated.

In scheduling problems that occur in practice it is often hard to make a good model that yields useful results. Furthermore in many cases the development and application of suitable solution methods is a difficult issue.

In this graduation project, for a particular practical scheduling problem at Prodrive Technologies we developed a model and applied several algorithms for computing approximate solutions. The algorithms described in this project are taken from the literature. These algorithms are applied to test and validation sets.

A substantiated and critical bench-marking approach is used to compare the implemented local search methods, neighborhood structure and meta-heuristic algorithms.

In Section 8.1 we answer the research questions that are considered in this graduation project. In Section 8.2 we summarize important conclusions that can be drawn from the performed experiments. After this, in Section 8.3 we give some recommendations for Prodrive Technologies. Finally, in Section 8.4 we give some suggestions for future research that can be performed to extend this study.

8.1 Research questions

In this section, we reflect on each research question of this graduation project as they are described in Section 1.5:

Research question 1: can a model-based scheduling process lead to significant improvement in the efficiency of the manufacturing process of the SMD PCBA manufacturing division?

A proof of concept is created that can directly be used by the SMD PCBA manufacturing division. This proof of concept uses a simple single line schedule and minimizes a sequence dependent weight. A greedy algorithm is combined with a tabu search algorithm. Positive feedback is received from the production floor and the application is integrated in the current scheduling process. Given that this proof of concept already led to an improvement and that the work of this thesis considerably improves upon the proof of concept, we conclude that it is indeed possible to use such a process to improve this efficiency.

Research question 2: which constraints and cost function are useful for modeling our practical problem?

Two constraints are identified that must be met by our method so that it yields feasible schedules. The first constraint is a release time for each job. Each job must be scheduled after this release time. The
second constraint is the possibility that a job can only be produced on a subset of the lines. These two constraints are met by the schedules that are determined by the implemented algorithms.

A cost function has been developed that contains the most significant costs that can be influenced by changing the schedule. The cost function consists of four parts, namely the changeover cost, feeder preparation cost, warehouse cost and tardiness cost. A concrete description of this cost function and a substantiation how it is retrieved from the data of the SMD PCB manufacturing division is presented in Chapter 3.

**Research question 3:** is it feasible to apply exact optimization methods to our practical problem?

In Chapter 4 a MIP formulation is given that can be used to determine the global optimum of the described scheduling problem. Furthermore a set partitioning formulation is given to which a column generation algorithm can be applied. It is expected that the largest problem instances that can be solved within a reasonable amount of time consist of 45 jobs. This problem size is significantly smaller than our practical problem at hand. Hence, we conclude that it is currently not feasible to apply exact optimization methods to our practical problem.

**Research question 4:** which heuristic algorithms can be applied for the approximate solution of our optimization problem, and how do these perform?

Experiments with several variations from four different classes of heuristic algorithms are performed. Based on a literature review we decided to restrict to these four classes. These classes are (iterated) local search, simulated annealing, tabu search and genetic algorithms.

The best performing algorithms implemented are simulated annealing and tabu search. Further comparisons are discussed below.

### 8.2 Further conclusions

In this section we discuss a few further conclusions.

The implemented variants of the class of genetic algorithms perform worst. We expect that this is because the practical problem does not fit well to the requirements for such algorithms. The crossover methods are expected to yield many unfeasible schedules which will result in a lot of evaluations of essentially random schedules.

An important observation made when experimenting with tabu search algorithms is that in general exact evaluation of the solutions in a full neighborhood of a schedule is not tractable. For the class of tabu search algorithms, the one using an approximate evaluation of the solutions in a neighborhood constructed by job jump mutations performs best.

The implemented variants of the class of simulated annealing algorithms perform well. The variants that use a job jump and job swap neighborhood structure perform better than the implemented k-opt neighborhood structures. Furthermore variants using an exponential cooling scheme perform better than the variants using a linear cooling scheme.

In our experiments we observe that the trend of the cost function shows a very different behavior if one considers strongly varying problem sizes. When considering problems of 100 jobs, finding multiple local minima is still tractable, while for problem instances of 400 jobs, after 6 hours still no local minimum is found.

In the results of the performed experiments, for test problem instances as well as for validation problem instances it can be seen that the expenses of the SMD production lines can be decreased significantly.
8.3 Recommendations

During this graduation project, many observations are made that may help to improve the production process. This section lists a few important recommendations in order to increase the expected business value for Prodrive Technologies.

First, however, we note that statistics about the quality of the optimized schedules using the recommended algorithm on the operations department have not been presented, since the algorithm is not introduced in this department, yet. The main reason is that currently short and long orders are split into two clusters that are planned on separate SMD production lines. Optimized schedules are reviewed with supply chain planning as well as shop floor planning. The schedules are considered practical as well as feasible and therefore can be used by the operations department. The most important consideration is that introduction of the new scheduling process mixes all orders over all SMD production lines, which will result in a new way of working on the production floor. Therefore this must be carefully prepared. We plan to introduce this in Q1 2020. We plan to start by scheduling the proto-orders using an implementation of the recommended meta-heuristic algorithm and then, step by step, include the longer orders so that operators can slowly adapt to the new situation.

**Recommendations:**

- **Fast optimization:** for a schedule that is directly needed, a local search algorithm that uses an approximate evaluation of a neighborhood constructed by job jumps is recommended. As a starting point for this algorithm the described greedy algorithm is recommended.

- **One week optimization:** for an optimization of the production schedule for a week, the tabu search algorithm that evaluates the approximate costs of a job jump neighborhood structure is recommended.

- **Two/four week optimization:** for a schedule optimization of two or four weeks, a simulated annealing algorithm using an exponential cooling scheme is recommended. For a neighborhood structure we recommend to pick solutions from the job swap mutation.

- **PNP group setup:** as described in the introduction it is possible to prepare group setup for the SMD production lines. These group setups are considered as input for this scheduling algorithm. Finding the best jobs to put into a group is a time consuming task. Furthermore evaluation of all possible groups can not be done by a single work preparation engineer. Therefore it is recommended to start a pre-study to investigate the existing algorithms for group setups so that this can be automated. It is expected that finding a good group setup proposal will decrease the cost significantly.

- **Software application:** to be able to easily move the responsibility of the scheduling process to production it is recommended to implement or buy a software application that can be used to perform the scheduling algorithm. An example of such an application can be found in Appendix C. The application can use automated hyper-parameter tuning based on the results of the data analysis.

- **Regular update of cost function:** It is recommended to perform the data analysis as described in Appendix A and update the parameters in the selected algorithm every 3 months. Furthermore it is recommended to do this when the machine environment or SMD process changes. Also in the analysis described in Appendix A only a single interval is considered. It may be wise to look at multiple instances, to see whether the measurements vary.

- **Improve expected production times:** it is recommended to investigate the possibilities for more accurate prediction of the production time of an order. Currently on the production floor there are still many orders (especially for new products) for which the production time is incorrect. This causes line switches which are expensive and often unnecessary.

- **Improve feeder preparation:** at the feeder preparation subdivision only a single tape per order and component is prepared in a feeder. If the order size requires a second tape, this tape is provided without a feeder to the SMD production line. This often results in an SMD production line standing still while an operator is binding in a feeder. We expect that it is more cost efficient to bind all tapes at the feeder preparation subdivision.
• **Downtime compliance**: the model developed in this project does not contain the notion of
downtime of a line for maintenance purposes. A known work around is extending the pre-
condition with this. For the new planning process proposed, this is inconvenient. We recommend
to investigate this further.

### 8.4 Future research

We briefly address a few topics that we consider to be of interest for further research. The goal of this
section is to steer a follow-up researcher in the correct direction.

The following topics could be investigated in the future. These research topics are all extensions that
can be based on the results presented in this thesis. We ordered the items below according to their
relevance for obtaining better results:

- **Combined neighborhood structure**: in this graduation project, four mutation methods that
create a neighborhood structure are studied. As briefly discussed to in Section 5.2, also
neighborhood structures can be composed by applying some of these mutation methods in
parallel and considering all resulting solutions as the neighborhood. Furthermore also other
mutation methods exist. A project that further exploits the local search options is expected to be
beneficial since it improves the basis for nearly all algorithms applied in this graduation project.

- **Multi-mutation neighborhood**: Increasing the neighborhood size by multi mutation steps
seems to be a very interesting issue, especially in combination with the fast approximate
neighborhood evaluation that we described. In this way, a significantly larger part of the search
space can be explored.

- **Exact method**: For the current methods it is still unclear how far we are from the optimum.
Even for small test cases the global optimum remains unknown. By implementing a MILP and
experimenting with test cases that have a small size, we can get more insight into the optimality
gap. Furthermore by performing a comparison of the schedules that are obtained by the exact
and heuristic algorithms one can study the structure of local minima.

- **Genetic algorithm**: The experiments performed using genetic algorithms do not yield good
results. This may be because this algorithm is not suitable for our problem. There, however,
is a demand for further research on, and experiments with, different crossover and mutation
methods.

- **Evolution strategy**: all performed experiments with the class of population based algorithms in
this graduation project implement the subclass genetic algorithms. There may be other algorithms
in the class of population based algorithms that work better for our problem. An algorithm that
seems interesting for further research is the covariant matrix adaptation evolution strategy (CMA-
ES). This algorithm is very popular nowadays, especially for continuous optimization problems.
Recently, more and more papers appeared that describe applications of this algorithm in the
field of discrete optimization.

- **Improve cost function**: In this thesis a cost function is proposed and tested that is custom for
our SMD PCBA production environment. This cost function is already closer to reality than the
cost model used in the proof of concept. A study concerning the quality of this cost function and
possible (fast) approximations of costs can be beneficial.
References


Appendix A

Data Analytics

This appendix chapter describes the data analysis performed during this graduation project. They are documents as substantiation for the described cost function in Chapter 3. Also this possibility of re-analyzing the same data in the future.

When analyzing data, it is crucial that the quality of data is accessed properly. A model of data quality metrics is normally used for this. The most common used data quality dimensions are: accuracy, validity, integrity, completeness, consistency and timeliness.

Since the goal of this analysis is to retrieve a total cost based on a number of occurrences especially the data accuracy metric is to be questioned. One of the obvious and hugely used multipliers for this is the median. Since in an analysis described in this chapter the median is pulled up by outliers in only one direction it is decided to use another method. The second method provided more realistic results and is therefore used in the cost function. It first filters the outliers from the sample data, followed by taking the mean of the resulting intervals.

For the filtering of outliers, multiple methods exist. Some commonly used methods are the standard deviation method, the interquartile range method (IQR) and DBSCAN [43]. Scatter plots of the required intervals indicated that the IQR method is sufficient and therefore this is used for all outlier filtering performed during this graduation project. The IQR method considers all values outside of two bounds ($b_l$ and $b_u$) an outlier. These bounds are calculated as follows: $b_l = Q_1 - 1.5 \cdot IQR$ and $b_u = Q_3 + 1.5 \cdot IQR$ where $IQR = Q_3 - Q_1$. For all performed analysis $b_l$ is negative and therefore no values lower than this bound can exist.

A.1 Data warehouse

The described cost function is based on average timings measured at the logistics- and SMD PCBA manufacturing division. These values heavily depend on the warehouse size, warehouse location, average operator experience etc. Therefore it is essential that it is possible to observe changes in the required data so that the cost function can be corrected accordingly.

The data that is required to calculate the averages used in the described cost function are retrieved from multiple software systems. These systems are SAP R/3 Enterprise [44], an in-house developed software application called the logistics execution application (LEA) and Fuji Flexa [45]. Using this data intervals must be calculated, preferably automatically. Furthermore this data may be useful for various purposes in the future (for example other process optimizations). Therefore these analysis should be setup in a generic future proof manner.
It was decided to prepare all data in an Microsoft Azure [46] structure that complies with the ETL process and creates a data warehouse that complies with the dimensional fact model. Based on this data warehouse Microsoft Power BI [47] is used to create interactive reports containing the required data. The scatter plots presented in consecutive sections are retrieved from these reports. This results in views being available to all interested parties within Prodrive. The data in these reports is updated by a nightly run and thus is at most one day old. Figure 1.1 shows the structure of the created data warehouse and the dependencies of the created reporting.

A.2 Feeder movement

As described in Chapter 1, the tapes that contain the components placed by the Fuji PNP machines are placed in Fuji feeders before they go into the PNP machines. Each of these feeders has its own unique identifier that it registered by a software application developed by Fuji called Fuji Flexa [45]. An in-house developed software application retrieves the database of Fuji Flexa periodically and keeps track of when feeders are placed and removed. This software application is called the PNP material tracer. Over the data that is stored by the PNP material tracer intervals are calculated per feeder. This is done in the transformation stage of the described data warehouse.

When analyzing the feeder intervals in the Power BI report, the right IQR bound of line to line transfers $b_r = 27.43$ seconds, the right IQR bound of line to feeder carrier transfers $b_r = 119.61$ seconds and the right IQR bound of carrier to line transfers $b_r = 906.78$ can be found. Limiting the scatter plot in the interactive Power BI reports allows finding an average time of 6.51, 27.56 and 165.64 seconds for transferring feeders from line to line, feeder carrier to line and line to feeder carrier respectively. These scatter plots can be found in Figure 1.2, Figure 1.3 and Figure 1.4.
Figure 1.2: A scatter plot of the interval duration of feeder moves from a PNP machine to a PNP machine performed at the SMD production environment during 2 regular production days.

Figure 1.3: A scatter plot of the interval duration of feeder moves from feeder carrier to a PNP machine performed at the SMD production environment during 2 regular production days.
A.3 Feeder preparation

At the feeder preparation subdivision tapes provided by the logistics division are placed in feeders. The placement of tapes in feeders is often called feeder binding. Also at this subdivision, tapes are removed from feeders before they are returned to the logistics division. When feeder binding or inbinding is performed on a feeder, this feeder must be placed into a loading unit. The moment in time that feeders are placed in or removed from a loading unit is traced by an in-house developed software application called the logistics execution application (LEA). Using the data stored by this application the intervals between feeder in- and outbind actions can be calculated, visualized and analyzed.

When analyzing the created Power BI report, the right IQR bound for feeder binding and tape remove actions can be found. The bounds are $b_r = 258.32$ and $b_r = 252.74$ respectively. Limiting the scatter plot in the interactive Power BI reports allows finding an average time of 62.94 and 51.17 seconds for feeder binding and tape remove actions respectively. These scatter plots can be found in Figure 1.5 and Figure 1.6.

Figure 1.4: A scatter plot of the interval duration of feeder moves from a PNP machine to feeder carrier performed at the SMD production environment during 2 regular production days.

Figure 1.5: A scatter plot of the interval duration of feeder binding actions performed at the SMD production environment during 15 regular production days.
Figure 1.6: A scatter plot of the interval duration of tape remove actions performed at the SMD production environment during 15 regular production days.

A.4 Warehouse

All tapes that are picked for a specific order at the SMD production environment are picked by a single employee at the logistics division. This warehouse employee uses a hand-terminal that is registered to the concerning employee to scan and book the tapes that are picked. These bookings are stored in a single transfer order. This hand-terminal registers the exact moment the tape is scanned in the concerning transfer order in SAP R/3 Enterprise [44].

The time between two measured of pick actions can be analyzed by calculating intervals between transfer order items within each transfer order. To only select the relevant data, all intervals touching or between transfer order items that are performed for other departments than the SMD production environment must be filtered out.

The transfer order items that concern a SMD production environment tape pick can be recognized in the following manner:

- The source bin had a "T" as prefix.
- The source bin has a ".-" as fourth character counter from the end.
- The destination bin does not start with a "T".

Similar as for pick actions, tapes are transferred back to stock in the warehouse per order. Also, this is done by a single warehouse employee that bookings tapes on a single transfer order. The transfer order items that concern a SMD production environment stock transfer can be recognized by the inverse properties of a pick actions.

When analyzing the created Power BI report, the right IQR bound for pick actions and stock transfers are $b_r = 42.5$ and $97.5$ respectively. Limiting the scatter plot in the interactive Power BI reports allows finding an average time of $15.28$ and $22.08$ seconds for pick actions and stock transfers respectively. These scatter plots can be found in Figure 1.7 and Figure 1.8.
Figure 1.7: A scatter plot of the interval duration of tape pick actions performed at the SMD production environment during 30 regular production days.

Figure 1.8: A scatter plot of the interval duration of tape stock transfers performed at the SMD production environment during 30 regular production days.
Appendix B

Pseudo-code algorithms

This chapter gives a pseudo-code representation of some of the algorithms implemented during this graduation project. Note that most pseudo-code procedures represent the structure of a class of algorithms. This means that it is representative for more than one of the implemented algorithms. The implemented algorithms are described in more details in Chapter 5.

Algorithm 1 Create random schedule

Require: \( M \) - a set of machines to schedule on
Require: \( J \) - the set of jobs that are scheduled
1: procedure CREATERANDOMSCHEDULE\((M, J)\)
2: \( s \leftarrow \text{SCHEDULESTATICJOBS}(s) \) \( \triangleright \) A list containing the remaining candidate jobs
3: \( C \leftarrow J \)
4: while \( |C| > 0 \) do
5: \( s \leftarrow \max \) \( \triangleright \) Target for the makespan of the earliest finished machine
6: for \( mc \in M \) do
7: \( j_{\text{last}} \leftarrow \text{LASTJOB}(mc) \)
8: \( C_j \leftarrow \text{COMPLETIONTIME}(j_{\text{last}}) \)
9: if \( C_j \leq s \) then
10: \( m \leftarrow mc \)
11: \( s \leftarrow C_j \)
12: \( j \leftarrow \text{GETRANDOM}(C) \)
13: if \( j_r > s \) then \( \triangleright \) Ignore early jobs
14: \( s(m, \text{end} + 1) \leftarrow j \) \( \triangleright \) Schedule the job
15: \( C \leftarrow C \setminus j \) \( \triangleright \) Remove the job from the set of candidates
16: return \( s \)
Algorithm 2 Create greedy schedule

Require: $M$ - a set of machines to schedule on
Require: $J$ - the set of jobs that are scheduled
Require: $X$ - a matrix with the changeover costs between all jobs

1:  procedure CREATEGREEDYSCHEDULE($M, J, X$)
2:     $s \leftarrow$ SCHEDULESTATICJOBS($s$)
3:     $C \leftarrow J$  \hspace{1cm} ▷ A list containing the remaining candidate jobs
4:     while $|C| > 0$ do
5:         $s \leftarrow \max$  \hspace{1cm} ▷ Target for the makespan of the earliest finished machine
6:             for $mc \in M$ do
7:                 $j_{\text{last}} \leftarrow$ LASTJOB($mc$)
8:                     $C_j \leftarrow$ COMPLETIONTIME($j_{\text{last}}$)
9:                         if $C_j \leq s$ then
10:                           $j_{\text{select}} \leftarrow j_{\text{last}}$
11:                               $m \leftarrow mc$
12:                                   $s \leftarrow C_j$
13:                                   $c_{\text{best}} \leftarrow \max$
14:                                       for $j \in C$ do
15:                                           $c_{\text{current}} \leftarrow X(j_{\text{select}}, j)$
16:                                               $j_r \leftarrow$ RELEASETIME($j$)
17:                                                   if $c_{\text{best}} > c_{\text{current}} \land j_r > s$ then
18:                                                     $j_{\text{best}} \leftarrow c_{\text{current}}$
19:                                                         $j_{\text{best}} \leftarrow j$
20:                                                             $s(m, \text{end} + 1) \leftarrow j_{\text{best}}$
21:                                                               $C \leftarrow C \setminus j_{\text{best}}$
22:                                                                   return $s$

Algorithm 3 Local search

Require: $M$ - a set of machines to schedule on
Require: $J$ - the set of jobs that are scheduled
Require: $X$ - a matrix with the changeover costs between all jobs

1:  procedure LOCALSEARCH
2:     $s \leftarrow$ CREATEGREEDYSCHEDULE($M, J, X$)
3:     $c \leftarrow$ EVALUATECOST($s$)
4:     while CANCELLATIONCRITERIA do
5:         $S_{\text{nh}} \leftarrow$ GETNEIGHBORS($s$)  \hspace{1cm} ▷ Various neighborhood structures can be used
6:                 $s_{\text{best}} \leftarrow \emptyset$
7:                     $c_{\text{best}} \leftarrow \max$
8:                         for $s_{\text{new}} \in S_{\text{nh}}$, do
9:                             $c_{\text{new}} \leftarrow$ EVALUATECOST($s_{\text{new}}$)
10:                                 if $c_{\text{new}} < c_{\text{best}}$ then
11:                                   $s_{\text{best}} \leftarrow s_{\text{new}}$
12:                                     $c_{\text{best}} \leftarrow c_{\text{new}}$
13:                                         if $s_{\text{best}} \neq \emptyset \land c_{\text{best}} < c$ then
14:                                             $s \leftarrow s_{\text{best}}$
15:                                               $c \leftarrow c_{\text{best}}$
16:                                                   return $s$
Algorithm 4 Local search (approximate evaluation)

Require: $\mathcal{M}$ - a set of machines to schedule on
Require: $\mathcal{J}$ - the set of jobs that are scheduled
Require: $\mathcal{X}$ - a matrix with the changeover costs between all jobs

1: procedure LOCAL SEARCH
2:    $s \leftarrow$ CREATEGREEDYSCHEDULE($\mathcal{M}, \mathcal{J}, \mathcal{X}$)
3:    $c \leftarrow$ EVALUATECOST($s$)
4: while CANCELLATIONCRITERIA do
5:    $S_{nh} \leftarrow$ GETNEIGHBORS($s$) ▷ Various neighborhood structures can be used
6:    $s_{best} \leftarrow \emptyset$
7:    $c_{best} \leftarrow \max$
8:    for $s_{new} \in S_{nh}$ do
9:        $c_{new} \leftarrow$ EVALUATEAPPROXIMATECOST($s_{new}$)
10:    if $c_{new} < c_{best}$ then
11:        $s_{best} \leftarrow s_{new}$
12:        $c_{best} \leftarrow c_{new}$
13:    if $s_{best} \neq \emptyset \land c_{best} < c$ then
14:        $s_{new} \leftarrow$ EVALUATECOST($s_{best}$)
15:    if $c_{new} < c$ then ▷ Verify if the cost actually improved
16:        $s \leftarrow s_{best}$
17:        $c \leftarrow c_{new}$
18: return $s$

Algorithm 5 Tabu search algorithm

Require: $\mathcal{M}$ - a set of machines to schedule on
Require: $\mathcal{J}$ - the set of jobs that are scheduled
Require: $\mathcal{X}$ - a matrix with the changeover costs between all jobs

1: procedure TABU SEARCH
2:    $s \leftarrow$ CREATEGREEDYSCHEDULE($\mathcal{M}, \mathcal{J}, \mathcal{X}$)
3:    $c \leftarrow$ EVALUATECOST($s$)
4:    $t_{size} \leftarrow 20$ ▷ This example uses a TABU list of size 20
5:    $T \leftarrow \emptyset$
6: while CANCELLATIONCRITERIA do ▷ Includes iterating TABU list
7:    $s_{new} \leftarrow$ GETHIGHESTNEIGHBOR($s, T$) ▷ Considers neighbors not in TABU list
8:    $c_{new} \leftarrow$ EVALUATECOST($n$)
9:    if $c_{new} < c$ then
10:        $s \leftarrow s_{new}$
11:        $c \leftarrow c_{new}$
12:    else
13:        $T(1 : t_{size} - 1) \leftarrow T(2 : t_{size})$
14:        $T(t_{size}) \leftarrow s_{new}$
15: return $s$
Algorithm 6 Simulated annealing

Require: $M$ - a set of machines to schedule on
Require: $J$ - the set of jobs that are scheduled
Require: $X$ - a matrix with the changeover costs between all jobs

1: procedure SIMULATED ANNEALING
2: $s \leftarrow$ CREATEGREEDYSCHEDULE($M, J, X$)
3: $c \leftarrow$ EVALUATECOST($s$)
4: while CANCELLATIONCRITERIA do
5:     $s_{new} \leftarrow$ GETRANDOMNEIGHBOR($s$) ▷ Pick a random neighbor
6:     $c_{new} \leftarrow$ EVALUATECOST($n$)
7:     if $c_{new} < c$ then
8:         $s \leftarrow s_{new}$
9:         $c \leftarrow c_{new}$
10:     else
11:         if $P(c, c_{new}, T) \geq \text{random}(0, 1)$ then ▷ Possibly accept worse solutions
12:             $s \leftarrow s_{new}$
13:             $c \leftarrow c_{new}$
14:     return $s$

Algorithm 7 Genetic algorithm

1: procedure GENETIC
2:   $P \leftarrow$ GETINITIALPOPULATION ▷ Including greedy schedule
3:   $s \leftarrow$ BEST($P$)
4:   $c \leftarrow$ EVALUATECOST($P$)
5:   while CANCELLATIONCRITERIA do
6:       $P_{selected} \leftarrow$ SELECTION($P$)
7:       $P_{crossover} \leftarrow$ CROSSOVER($P_{selected}$)
8:       $P_{perturbation} \leftarrow$ PERTURBATION($P_{selected} \cup P_{crossover}$)
9:       $P \leftarrow P_{selected} \cup P_{crossover} \cup P_{perturbation}$
10:      $s_{new} \leftarrow$ BEST($P$)
11:      $c_{new} \leftarrow$ EVALUATECOST($s_{new}$) ▷ Keep track of the schedule with the lowest cost
12:      if $c_{new} < c$ then
13:          $s \leftarrow s_{new}$
14:          $c \leftarrow c_{new}$
15:      return $s$
Appendix C

Example scheduling application

As previously mentioned, the goal of this assignment is to create an optimization algorithm that can optimize a job schedule automatically. This chapter gives an example of how the created optimization algorithm can be used in a proof of concept or in application developed by Prodrive in the future.

Since there may be a prior setup and tapes are picked prior to the start of production, orders can be locked. Furthermore it must be possible to schedule high priority orders at a fixed time and line. An example of how a possible schedule can look like, what is optimized periodically, and what stays on the same line, is shown in Figure 3.1.

The jobs that are being optimized periodically are optimized using an multidimensional optimization algorithm that approximates a minimum for the following cost functions:

- **total time**: the total time for a set of orders should be minimized, so that more can be produced in less time.

- **dependency risk**: for example, two jobs that need the same tape on a different line introduce the risk that if one order delays, the other line also delays because it needs this tape. These types of risk need to be avoided.
• **operator work**: the amount of work to prepare orders must be minimized, for example: when two orders are scheduled, that need the same tapes on another line. Then all tapes need to move to another line, which is a lot of work for the operators, it would be better to produce them on the same line.

• **workload variance**: the variance in workload on a line over time should be minimized such that operators can continue doing something and there are no peaks, since these will probably lead to downtime / inefficiency of the line.

• **workload peak**: if all lines need setup at the same time, the setup times will be longer because there are not enough operators at that moment, meaning that at other points in time they are waiting for the SMD production lines to finish. This must be avoided.

Note that these optimization goals are just an example, as described in the next chapter, a big part of the assignment is to investigate the correct optimization goals / cost functions.
Appendix D

Theoretical and practical research

This chapter describes the existing methods, heuristics and meta-heuristics that may be applicable to the described scheduling problem. Furthermore some existing experimental practical research and research that compares heuristics is described. Since the design-space of meta-heuristics for the describe scheduling problem is so huge, numerous papers describing research of applications in practice exist.

Literature to surface mount technology (SMT) scheduling optimization may focus on multiple aspects of the SMT production process. Many of those sub problems are not relevant for the described scheduling problem. [48] gives an elegant specification of the different sub problems. Partition (SP3) and sequencing (SP4) are the sub problems that the described scheduling problem covers. Therefore, papers about practical research of all other SMT production process sub problems are ignored.

Furthermore a lot of scheduling problems exist in the same practical field. None of the found papers covers the exact same theoretical scheduling problem but approaches for similar scheduling problems are considered interesting.

In the existing papers about research performed in this field tardiness is a common objective. Therefore, there is common ground between most of the scheduling problems studied in existing research and the described scheduling problem. Changeover time can be found less often in theoretical research but enough practical research exists.

The focus of this research is placed on the meta-heuristics, since the time complexity of the described problem in combination with the maximum instance size is expected to high to solve optimally using integer programming methods. Furthermore some basics are described about integer programming methods that may come in handy when an optimal reference is needed for small instances.

As can found during research and also can be found in [49] there are many meta-heuristics developed based on semaphores in the last few years that are not always properly substantiated. This leads to a lot of flawed conclusions that can be found spread over numerous papers. This makes research in this field tricky and asks for a critical eye.

This chapter first describing some well known integer programming methods that may be applied to find the optimum solution to some very small problem instances in Appendix D.1. These methods can be used to find an optimal solution that can be used as a reference to find the optimality gap of non optimal methods. Secondly the possible local search methods are described in Appendix D.2. And after that, in Appendix D.3 known meta-heuristics are described.

D.1 Integer programming methods

Integer programming methods can be used to solve different type of linear programming problems. This section describes some well known existing methods that can be used to solve these kind of problems. Furthermore some practical applications are mentioned.
Integer programming is one of the 21 problems for which Karp defined that they are more complex than the boolean satisfiability problem (SAT) [6]. Therefore, since SAT is NP-complete, linear integer programming is NP-hard.

The described scheduling problem can be formulated in a non-linear method. Therefore, before linear programming methods can be applied, it first must be linearized. This can for example be done by adding variables to split ‘or’ cases.

If the described scheduling problem is declared in a linear form, the simplex method (also called Dantzig’s simplex algorithm) can be used to solve it. If calculating an optimal solution using the simplex method is not tractable, (delayed) column generation can be considered. Column generation is a method that can be applied to generate lower and upper bounds when applying the simplex method. It is based on an extra action performed during the pricing phase of the simplex method.

The branch and bound algorithm decreases the problem size by pruning branches based on upper and lower bounds found in other branches (for example by finding a feasible point or LP relaxation). A combination of (delayed) column generation and the branch and bound algorithm can be used to tackle the described scheduling problems in linear form. Also this approach gives an optimal schedule if it is tractable. For complex problems in combination with large numbers of jobs this may not be tractable. The method can also be partially calculated and used as a heuristic. This can be done by stopping it after a specified number of iterations and using the best case as lower bound.

In [7] a near optimal solution is presented based on Lagrangian relaxation for the theoretical scheduling problem $P|p_j|\sum T_j$. Note that this problem is also NP-hard. It generates a lower bound on the cost. This can be used to measure the level of sub-optimality. It is stated that they reach within 1% of optimal for most schedules within reasonable computation time.

In the paper [8] the column generation algorithm is used on a parallel machine problem in 2010. The three-field notation of the problems that where investigated are $P|\text{divides}|L_{\text{max}}$ and $P|r_i;\text{prec}|L_{\text{max}}$. These problems can be considered simpler than the problem that is coped with during this project. It was observed that the performance increased by the number of jobs per machine (significantly starting from 20 jobs per machine).

### D.2 Local search

Local search is a method that can be used to find good solutions for problems by traveling through the solution space. It is almost always applied on problems for which an exhaustive search is not tractable because they have a to high time complexity. This section describes the basic local search terminology and describes how it can be applied on the described scheduling problem.

The basic idea of a local search algorithm is to step from a feasible solution to one of its neighbors in an improving manner. A neighbor of the feasible solution is often called candidate solution and the set of candidate solutions is called the search space. Local search is a basic heuristic that is commonly used by most of the later described meta-heuristics. It is essential to select the best local search method for the described scheduling problem, since improving by the use of meta-heuristics introduces significantly more time complexity than improving the local search method.

Neighbors of solution instances are easiest detected using graph theory, therefore a single solution instance of the described scheduling problem is first described as a graph:

A single solution for the described scheduling problem can be mapped to a graph in the following manner. Let every job in the bucket be a vertex and every changeover be an edge. In this case we are looking for a path cover of a complete un-directed graph with at most $m$ paths. The earlier described changeover work effort and changeover time are weighted on the edges. Furthermore the tardiness is also a weight on the edges, but note that changing an edge can change the cost of sequential selected edges.

One of the most commonly used neighborhood selection technique is 2-opt, which was first proposed by P. Croes [5]. This neighborhood selection technique removes 2 edges from a path followed by reconnecting the path by cross connecting the disconnected vertices. If the new solution is feasible, a new candidate solution is found. A more general form of 2-opt is k-opt. Using this neighborhood
selection technique, k edges are disconnected instead of 2. Note that the number of ways the vertices can be reconnected increases significantly when k increases.

K-opt can also be applied on edges of multiple paths. An important observation to take into consideration is the cost introduced by the tardiness of all vertices. This cost changes significantly when connecting the end parts of two different initial paths. In this case, one end path has to flip its ordering, which will probably never result in a good candidate solution.

### D.3 Meta-heuristics

This section describes some existing meta-heuristics and how they can be applied on optimization problems. Furthermore some references to existing research of related practical applications is given and relations to the described scheduling problem are described.

A meta-heuristic is a heuristic on a higher level than for example local search. There are a lot of meta-heuristics and therefore a good categorization is needed. J. Dreo made such a categorization [50] and this categorization can be found in Figure 4.1.

![Figure 4.1: The different classifications of meta-heuristics represented in an Euler diagram.](image)

A guided local search builds up a penalty during the search for solution instances where close local minima are found. This penalty is included in the cost function so that these local minima are avoided and thus better minima can be found. Guided search can first be found in a paper by C. Voudouris [51].

The tabu search algorithm is a deterministic algorithm that builds up a list of solutions that are already evaluated. This list of solutions is used to avoid solutions when searching for new candidates. Since the tabu algorithm uses steepest descent with exception from the tabu list, the candidate with the highest edge that does not lead to a solution in the tabu list is selected. When the tabu list is full, and a new local optimum is found it is added to the tabu list, another solution must be removed. Items are removed from the tabu list in FIFO order. The tabu algorithm was created by Fred W. Glover [4].

Simulated annealing is a meta-heuristic that changes during the traversal of the search space from stochastic to more and more deterministic. The basic principle is that it finds a common high area...
so that it can find a good local optimum and maybe even a global optimum when more and more
determinism is used. An advantage of the stochastic of the algorithm is that it can escape local
minimums. This algorithm has been discovered by S. Kirkpatrick, C. Gelatt and M. Vecchi in 1983 [21].

Variable neighborhood search is a stochastic meta-heuristic that repeats the selection of a random
neighbor and performing a local search from that solution until a termination condition is met. The
algorithm is based on three principles: a local minimum of one neighbor is not necessarily a local min-
ima for another neighbor, a global minima is a local minima with respect to all possible neighborhoods
and local minimum are for many problems close to the global minimum. The alorihm can first be
found in a paper by Mladenovic Hansen [52]. There are a lot of variations of the variable neighborhood
search algorithm, [53] gives a clear description of the different variants.

The ant colony optimization algorithm finds a Hamiltonian path in a graph. It solves for example the
traveling salesman problem. Since it does not implement iterative improvement by going through the
design space between iterations, the start and end will never change. Therefore, it is not expected to
find a global optimum for our problem. It is proposed in a paper by M. Dorigo in 1991 [54].

The particle swarm optimization algorithm maintains a set of solutions for which each solution has its
own direction and velocity. Each iteration the direction of each particle is update based on its own
direction, the direction of the best point and the direction of the previous best point. The basic idea is
that more and more particles move to area of the search space were good solutions can be found.
Partial swarm optimization is a population based algorithm that is initially proposed by J. Kennedy
and R. Eberhart [55].

Scatter search is a population based algorithm proposed by F. Glover [56]. A scatter search is build
up out of four steps that are repeated until a cancellation criteria is met. The steps are reference set
update, subset generation, solution combination and improvement method. Each step is executed
over the population and can be implemented in numerous ways and for the improvement method,
commonly a local search is used.

Genetic algorithms are population based algorithm and are evolutionary. It constructed out of three
operations namely selection, crossover and mutation. These mutations are repeatedly and consecu-
tively performed in the given order and each of the operations changes the population. Each of these
operations can be implemented in numerous ways. Picking these operations poorly will probably
intensificate the population to a subset whereof all points are close to each other in the search space.

F. Werner gives an overview of some possible application of genetic algorithms on (open, job and flow)
shop scheduling problems [31]. Examples of an application of the three different operations to similar
problems can be found here. Furthermore in the conclusion of the paper, it can be found that the often
slow convergence of genetic algorithms can be tackled by using heuristics that can be found in the
investigated types of scheduling problems.

M. van de Ven proposed a combination of simulated annealing and variable neighborhood search
[57]. Experiments are performed on data of the same target production environment that shows that a
slightly better objective can be reached compared to a normal variable neighborhood search and an
iterated local search. A big disadvantage is the heavy impact on the computation time.

In 'Scatter Search vs. Genetic Algorithms’ [58] scatter search and genetic algorithms are compared.
One of the studied problems is the traveling salesman problem, where the described scheduling
problem is related to. In this paper, a genetic algorithm and a scatter search algorithm are performed
on 31 problem instances and the outcome is that the scatter search algorithm performs significantly
better.

In 'Mm resource-constrained project scheduling by ACO with dynamic tournament strategy’ [59] exper-
iments are described that compare the ant colony optimization algorithm with simulated annealing
and a genetic algorithm on the multi-mode resource-constrained project scheduling problem (MRCPSP). In
the measurements based of the experiments it can be found that the ant colony optimization algorithm
performed better.