Floater Allocation in Paced Mixed-Model Assembly Lines

Design and implementation of a method for operational scheduling of cross-trained workers in Mixed-Model Assembly Lines

Master Thesis
Tjarco van Overbeek
May 2015- December 2015

Supervision University of Twente
1st supervisor: Dr. Ir. J.M.J. Schutten
2st supervisor: Dr. Ir. M.R.K. Mes

Supervision Ergo-Design
1st supervisor: Ir. P. Hentschke

Faculty & Educational program
Faculty of Behavioral, Management and Social Sciences
Master Industrial Engineering and Management
Track Production and Logistics Management

Date: 07-12-2015

UNIVERSITEIT TWENTE.

Summary

This thesis considers a paced Mixed Model Assembly Line (MMAL): an assembly line that handles multiple configurations/models of a product (cars in this thesis) that are transported with a constant speed. The time each station has to finish its activities is referred to as the takt of the line. Workers move alongside the assembly line while working on the product.

In these MMALs, stations see varying workloads. In the search of a high level of utilization, MMALs are often designed such that the average processing time at each station is approximately equal to the takt (Scholl, Kiel, & Scholl, 2010). This results in situations in which station cannot finish its activities on time. Workers can move out of the regular station bounds to keep working on the car. However, the distance that a worker can move out of a station (called overlap) is limited, due to availability of parts and/or power tools. When the car reaches the end of the overlap, the worker is forced to move back towards the beginning of his station to start working on the next car. The amount of seconds that a station falls short in completing its activities is referred to as *overtime* in this thesis, and every occurrence of overtime counts as one *defect*.

The client (referred to as Company A), has indicated that line stops and lowering the feeding/transportation pace of the assembly line are not desired options to resolve these situations. This is where floaters step in. Floaters are cross-trained workers who provide temporary support to 'busy' stations. In the current way of working, floaters are employed reactively, responding to requests for help from stations. This results in sub-optimal usage of floaters, missing opportunities to resolve overtime. Therefore, the goal of this thesis is:

To develop a tool for the operational scheduling of floaters in the assembly line of Company A, with the goal of minimizing overtime and defects in the final assembly.

We created a model that uses Car/Station Combinations (CSCs) as basis. Each CSC 'occurs' in a fixed timeframe: from the moment the car enters the station until it leaves the station. Help can only be provided by floaters within this timeframe.

The assembly line is divided into working areas that each have an assigned number of available floaters. We create schedules for each working area separately. Our proposed solution method first creates an initial solution using a Parallel First Come First Serve heuristic, resulting in a schedule for all available floaters. This initial solution is then further optimized, by iteratively taking out small parts of the floater schedule in which problems occur, called partial schedules, and optimizing them using Simulated Annealing. The quality of a floater schedule is determined by calculating the objective function with the form $(w1 * \sum Overtime + w2 * \sum Defects)$.

We implemented the solution method and carried out a numerical analysis to determine the optimal settings for the parameters of the methods.

For the total assembly line, an average reduction in overtime and defects of 84% and 79% respectively was achieved compared to the current way of working.

Foreword

Before you lies my master thesis, which marks the end of my master's programme Industrial Engineering and Management ('Technische Bedrijfskunde' in Dutch). In this thesis, I developed a method to schedule cross-trained workers, called floaters, in a paced assembly line. Floaters provide support to stations along the assembly line that are too busy. The challenge is to schedule floaters such that they resolve as much problems as possible. In the scheduling process travel times have to be taken into account, as the assembly line considered in this thesis covers a total area of 300 meters long and 200 meters wide.

I have enjoyed working on this assignment for several reasons. The assignment was challenging, based on a real case, and gave me the opportunity to walk through a complete process; from formulating a model and solution methods to evaluating and optimizing the performance of those methods.

This thesis would not have been possible without the help of several persons. First, I thank the persons at Ergo-Design, the providers of the assignment. Paul Hentschke acted as my external supervisor, and was always open to talk about the assignment. The basis of this assignment is a model, and one can sometimes forget what a model represents: reality. Paul helped a great deal in making sure the final solution method would be useful in practice. I also thank the owners of Ergo-Design, Jaap Westerink and Douwe Bonnema, for giving me the opportunity and the resources to make this assignment possible. Their positivity and flexibility made working at Ergo-Design a very pleasant experience.

Next, I thank my supervisors at the University of Twente, Marco Schutten and Martijn Mes. I enjoyed the mix of criticism and positive feedback during our meetings. Considering the number of graduate students they were supervising at the same time, I was surprised by the level of thoroughness by which they reviewed my work. The feedback and insight provided by Marco and Martijn helped to improve the quality of this thesis.

Finally, I thank my family, girlfriend, friends and roommates for their support. Especially those who had to put up with me rambling on about my assignment. If I mistakenly thought that you were still interested after the first minute, I apologize.

I hope you enjoy reading this thesis as much as I enjoyed writing it.

Table of Contents

1. Introduction

This thesis was written as part of a graduation internship at Ergo-Design B.V. in Enschede. This company is currently developing a simulation based day-to-day planning system for the final assembly at a car manufacturer that we shall refer to as Company A in the remainder of this thesis. Any (information or) images used in this thesis are chosen in a way that conceals the company's identity. The final assembly at Company A is done on an assembly line that can handle multiple models of cars with different configurations, called a mixed-model assembly line (MMAL). The goal of this graduation assignment is to contribute to the planning system, by developing a tool for scheduling cross-trained workers (called floaters) to assist regular workers on the assembly line. The model is being developed for a single plant, but the possible deployment of the package at other facilities has been taken into consideration by Ergo-Design.

In this chapter, an introduction is given to Ergo-Design and (mixed-model) assembly lines in Section 1.1 and 1.2 respectively. Then the challenges that arise on these assembly lines are introduced in Section 1.3. Possible methods of dealing with these challenges are discussed in Section 1.4. In Section 1.5 the problem is described and the scope is defined, resulting in the research plan in Section 1.6.

1.1 Ergo-Design B.V.

Ergo-Design is a consultancy company that specializes in Industrial Engineering. It was founded in 1991 and has 14 employees. Its operating areas can be grouped by factory design, lean production, and logistics and planning as shown in Figure 1. Activities of Ergo-Design consist of developing master plans for production facilities and logistic networks, and designing plant and production line layouts. Other activities include optimizing efficiency, logistics, quality, and ergonomics, performing simulation studies and developing smart scheduling applications such as takt clocks and digital scoreboards. References (among many others) include Scania, Volvo, Volkswagen, NAM, Akzo-Nobel, Tata, Thales, and Bosch.

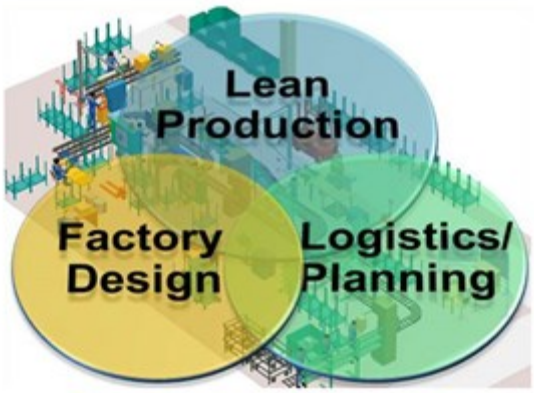


Figure 1. Operating areas Ergo-Design B.V.

1.2 Company A

Company A is a car manufacturer that uses an assembly line for the final assembly of their cars. An assembly line *is "a flow-oriented production system where the productive units performing the operations, referred to as stations, are aligned in a serial manner"* (Boysen, Fliedner, & Scholl, 2007). The assembly line at Company A can assembly different models and configurations of cars on the same line, with batch sizes as small as one and negligible set-up times in between. This is called a Mixed-Model Assembly Line (MMAL). MMALs provide several challenges (discussed in Section 1.3), and to cope with these challenges Company A has decided to develop a day-to-day planning system together with Ergo-Design. This planning system will simulate production runs based on input data such as which cars to produce on a certain day. The model will then provide processing times and other data that can be used to make an efficient production plan for that day. In this thesis we develop a tool that will be added to the total planning system. The tool will be used to schedule cross-trained workers to assist regular workers on the assembly line in an optimal way.

MMALs are employed in many production industries where flexibility is required. In the light of the problem at hand, we mainly refer to the application in the automotive industry in this thesis.

1.3 Challenges in MMALs

In Mixed Model Assembly Lines, products require different processing times on each station depending on their specification. In order to achieve an acceptable level of utilization, the line is designed such that the average processing time (at each station) is approximately equal to the takt of the line (Scholl, Kiel, & Scholl, 2010). As a result, labor intensive ('hard') products will have processing times longer than the takt at certain stations, whereas those of 'easy' models will be shorter. An assembly line is called *balanced* if the average processing times on all stations is approximately the same.

On the one hand, production companies aim for a high utilization to keep costs low. On the other hand, a high utilization means that imbalances on the line (differences in processing times) will lead to capacity *overload* situations, where a station cannot finish the product within one takt. The extra time the station needs to finish its activities is called *overtime*. Each occurrence of overtime is called a *defect*. Sometimes overtime can be resolved by regular workers by walking along the line outside the bounds of their working area. The distance that workers can walk outside their working station is called 'overlap'. This distance is limited, due to availability of power tools and limited working space. If the overtime cannot be resolved, then the work that remains unfinished has to be repaired either along the line (at a touch-up station) or after the product leaves the assembly line. These repair activities are costly as they require extra personnel and processing time. The main objective is to acquire a high utilization without deteriorating the assembly line performance in terms of quality and costs resulting from rework.

1.4 General solutions

In order to face the challenges mentioned in Section 1.3, imbalances on the line have to be prevented or compensated. There are several ways to do this.

The first option is called *line balancing* (Ghosh & Gagnon, 1989). Each station has a set of tasks assigned to it, such as installing a sunroof or an automatic transmission. Line balancing focusses on allocating these tasks to stations in a way that minimizes capacity overload situations by dividing workload as equally as possible (see Section 3.2). This process is generally done as part of medium or long term planning as it can involve relocating power tools and personnel or redesigning working areas. Inputs for line balancing are predicted product demand, the precedence relations of tasks and their processing times. As a result of changing demand, variations in working speed, defects in parts or malfunctioning power tools, it is hard to perfectly balance a mixed model assembly line. Line balancing cannot be carried out every day, and peaks in workload will remain that have to be resolved using short-term oriented methods.

The second option is *product sequencing* (Ghosh & Gagnon, 1989), where the sequence of products to be sent down the line is chosen (see Section 3.2). Given the allocation of tasks determined by line balancing, the processing times of cars on each station can be determined. By cleverly constructing a sequence, the resulting workload can be leveled for each individual station to a certain degree. For example, cars with a large workload at a certain station can be followed by one with a small workload at that station, to give the station time to 'catch up'. This is a simplified view on product sequencing, as the effects of changing the sequence become less clear as the number of stations increases.

A final approach is to temporarily change the available capacity at a station, by giving stations more time to finish activities, or increasing the workforce at a station. Stopping the line in case of an overload situation gives the station time to finish its activities, at the costs of the whole line having to wait. By changing the line speed, or the speed at which products are put on the line, the time available for stations can also be increased. The final option is temporarily increasing the workforce at a station. This can be done by assigning a floater (a cross trained worker) that will help the station when it is too busy.

1.5 Problem and Scope

Revisiting the previous section, we now describe the scope of this research and the reasoning behind it. The planning system developed by Ergo-Design is aimed at day-to-day planning. Therefore line balancing has fallen out of scope as this is a medium/long term activity that should not be carried out daily. The next step is product sequencing, which has already been carried out and implemented in the planning system (Op Den Kelder, 2015). After line balancing and product sequencing, overload situations still occur on the assembly line. Line stops and altering line/feeding speed are options that come at the cost of productivity, which is undesirable and has been indicated by Company A as such. The last remaining option to solve overload situations is the employment of floaters, which is the focus of this research (see Figure 2).

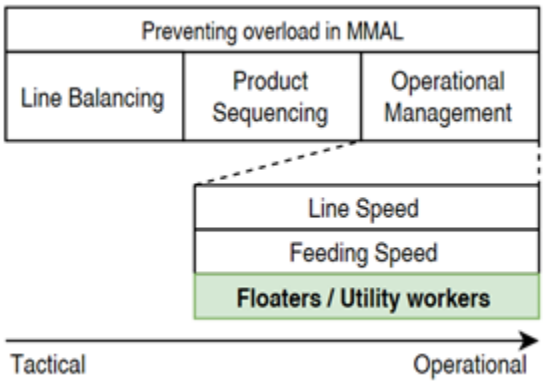


Figure 2. Research scope

The scope of this research is thus limited to the scheduling/allocation of floaters in the final assembly of Company A. It is assumed that required materials are at the right place at the right time. We consider the allocation of tasks to each station and their duration as given, in compliance with the planning system that has been developed so far. In this thesis, the sequence of cars is considered as given.

1.6 Research plan

We have discussed the general context and problem(s), and we have defined the scope of the research to be the scheduling of floaters in the final assembly. Currently floaters are already being used by Company A but in a reactive way without planning beforehand (see Section 2.3). Company A feels that they would benefit from an operational planning for floaters, and therefore the goal of this research is:

To develop a tool for the operational scheduling of floaters in the assembly line of Company A, with the goal of minimizing overtime and defects in the final assembly.

In order to successfully do this, we need to answer the following research question:

What is the best way to allocate floaters in a Mixed Model Assembly Line, in order to minimize overtime and defects?

We divide this question it into sub questions and answer them one by one. First, we must learn more about the assembly line (configuration, properties, etc.) and how floaters are scheduled in the current situation. Also, more knowledge of the planning system that has been developed so far is needed, in order to come up with an appropriate solution. In Chapter 2 we tackle the first research question:

1. What does the final assembly at Company A look like?
   1.1 What is the configuration of the assembly line?
   1.2 What are its properties?
   1.3 How are floaters scheduled in the current situation?
   1.4 What are the contents of the planning system and how does it work?

This step is taken before the literature review, as we do not need specialized knowledge to describe the current situation, and more insight in the production process allows us to review literature in a more focused manner. We review literature in Chapter 3 to see what earlier work has been done in relation to the scheduling of floaters, and what methods have been or can be used to solve this problem.

2. What methods can be used to schedule floaters in MMALs according to the literature?
   2.1 What is a Mixed-Model Assembly Line?
   2.2 What is the role of floaters in MMALs?
   2.3 How has the scheduling of floaters been dealt with in the literature?
   2.4 Which problems related to floater scheduling are known in the literature, and what methods can be used to solve them?
   2.5 What other methods can be used to solve the optimization problem in this thesis?

With the knowledge we have gathered from previous research questions as input, we can now construct a solution method in Chapter 4, answering the following research question:

3. What method is best suitable for minimizing overtime in the assembly line of Company A?
   3.1 What are the functional requirements to the solution design?
   3.2 What are the technical aspects of the problem?
   3.3 How should the current situation be translated into a model?
   3.4 What methods should be used to solve this model?
   3.5 How can the solutions be validated, e.g., how do we assure feasibility?

Now we have determined the method(s) that will be used for solving the floater scheduling problem, we implement them in the current planning system. Next, their parameters must be set, and the performance of the proposed solution method must be tested. Therefore we answer the following research questions in Chapter 5:

4. What are the optimal values for the different parameters of our methods, and how does the total solution method perform?
   4.1 What plan should we use to set the different parameters?
   4.2 What are the best settings for the parameters used by our solution method?
   4.3 What can be used as benchmark for the performance?
   4.4 What is the performance of the total proposed solution/method?
   4.5 How sensitive is the provided solution method to changes in key variables?

We discuss our methods and results, draw conclusions regarding the main research question, and give recommendations in Chapter 6.

2. Current Situation

In this chapter, a detailed description of the final assembly process at Company A is given. This answers the first research question, provides insight in the assembly process, and enables us to focus on relevant literature in Chapter 3.

2.1 Assembly line configuration

The production process under consideration is the final assembly at Company A. Cars are assembled on an assembly line, where workers use tools to perform manual jobs, such as installing the engine or doors. It is a mixed-model assembly line, meaning that different versions of cars can be produced on the same line, with virtually no set-up times in between. The assembly line consists of a total of 15 production lines that have a total of 222 workstations. It is a labor intensive process, and in total there are 343 regular workers active in the final assembly. A detailed subdivision is given in Table 1.

Line	# of workstations	# of fixed workers	Assembly examples
Body zone line	6	12	Tank and locks
Skillet line 1	21	28	Hood, suspension
Skillet line 2	21	40	Harness
Skillet line 3	23	30	Alarm, side windows, and bumper
Skillet line 4	20	38	Gear shift and handbrake
Window line 1	4	6	Windows
Chassis line	71	94	Fuel line, exhaust, radiator, oil reservoir
Engine line	18	25	Engine
Double Door Line	10	12	Doors
Door Line	4	8	
Subassembly line 1	24	50	Door speakers, door, mirrors
Subassembly line 2,3,4 & 5	Are part of the final assembly, but work without takt times and therefore fall outside the scope		
Total #	**222**	**343**	

Table 1. Elements of assembly line at Company A

A ‘car’ that enters the assembly line is a frame that somewhat resembles a complete car. An example of such a car frame is given in Figure 3.

Figure 3. Car frame

Before a car can be put on the road, a lot of components need to be installed. For nearly every component there are a number of options available. For example, a car can be bought with an optional air conditioning or sunroof. In this production process, the configuration of each car is stored in a so called '80 column row'. Each column represents an option, and the character in that column describes which option has been chosen for this specific car. An example of the 80 column row is given in Figure 4.

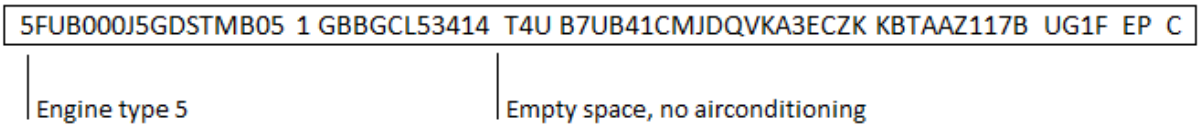


Figure 4. Example '80-column-row' for options

The amount of possible options per station differs from a minimum of one option to a maximum of 55 options. The current average number of options per station is 16.4. It becomes clear that a large number of different configurations are possible. The workload at a station for a given car depends on the options that are chosen. For example, a station installing the sunroof will have no workload if this option has not been chosen.

In this thesis we make the assumption that all parts are available for each car to be assembled. In reality, this assumption may not hold as suppliers might fail to deliver on time or certain parts have defects. Cars for which certain parts are missing will be removed from the list of cars that are available for production. Cars are selected from this list to enter the final assembly, and therefore cars will only enter the assembly line if all parts are available for assembly.

In Figure 5 a layout of the total final assembly is shown. Cars are stored in the storage tower before entering the final assembly. The cars are retrieved from the storage tower using an AS/AR system, and then flow through the assembly line following the arrows as shown in Figure 5. The sequence in which the cars are selected from the buffer tower is also the sequence in which the cars will flow through the assembly line, and stays the same for the entire process. The cars move through the body, skillet and window lines before entering the chassis line. The engine line produces engines parallel to the chassis line, and the engine is merged with the car at a certain point in the chassis line (indicated with a circled number one in Figure 5). After the chassis line, the cars enter the double door line. This line consists of two parallel lines, which are supplied with doors that come from the (single) door line. The double door line is the last line that falls within the scope of this research. A simplified scheme of the final assembly is shown in Figure 6.

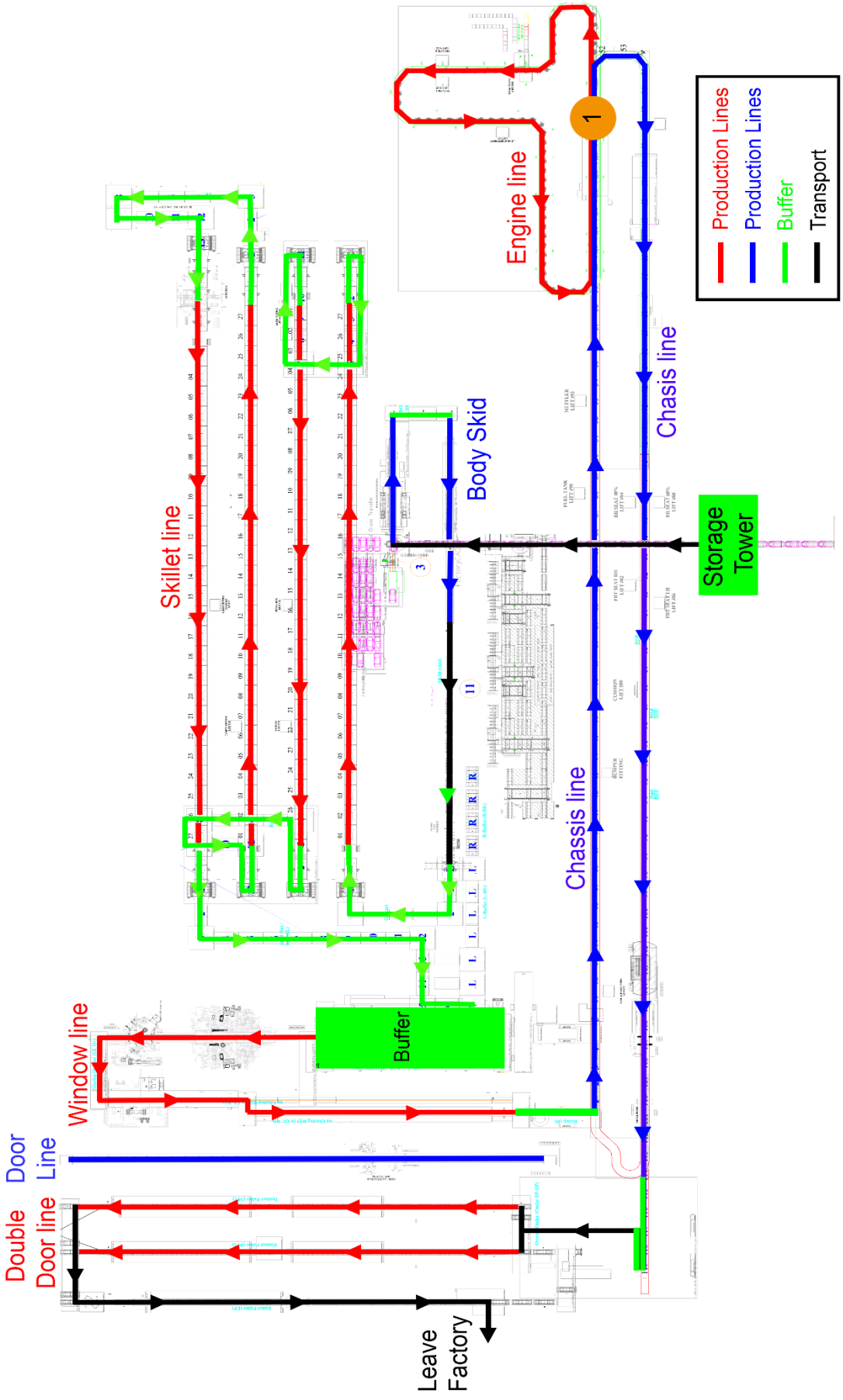


Figure 5. Schematic overview of the final assembly line at Company A

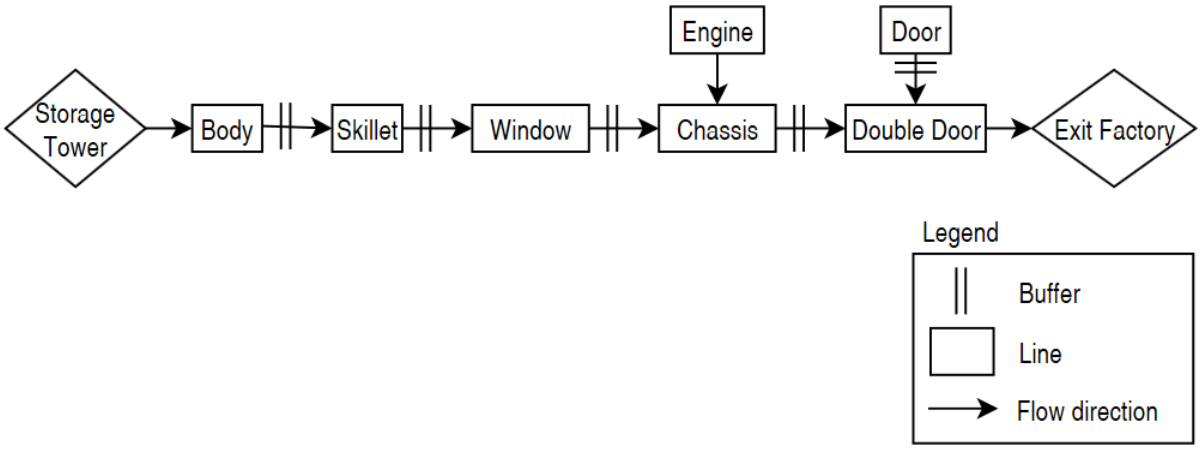


Figure 6. Assembly line flowchart

2.2 Assembly line properties

In Section 2.1 the general characteristics and layout of the assembly line were discussed. In this section, the detailed properties of the line are discussed.

Takt

In this assembly line a takt time is used of 52.8 seconds. Stations must finish their activities within this timeframe. Stations have a length between 3.5 and 7.5 meters, and the line speed at these stations is such that the car will leave the station after 52.8 seconds. Stations have different lengths because of requirements for room for tools and personnel.

Storage tower and buffers

The storage tower that precedes the final assembly has a maximum capacity of 730 cars. Based on a takt time of 52.8 seconds, the line is able to assemble 68.2 cars per hour. The maximum number of cars that can be produced in an 8 hours shift is 545, and there are 2 shifts per day.

Buffers have been placed between the different lines indicated with the color green in Figure 5. These buffers can be seen as empty stations, and cars will flow through them in the same rate (one per 52.8 seconds) as through the stations. These buffers can be seen as decoupling points in the assembly line, meaning that any overtime from the stations before the buffer have no effect on the stations after the buffer. This would only be possible if a worker would walk along the line through the whole buffer to complete his activities, which is realistically not possible because the 'walk-along' distance (or overlap) is limited to a distance that is much smaller than the buffers.

Lines, Stations and Workplaces

Each line (e.g. Chassis line, Body line etc.) consists of multiple stations, each having their own working area (length). Each station has 9 positions (three rows of three) that can be occupied by workers to perform assembly operations, called workplaces. There is a row to the left of the conveyor, one in middle and one to the right. In Figure 7 this architecture is shown for an example skillet line.

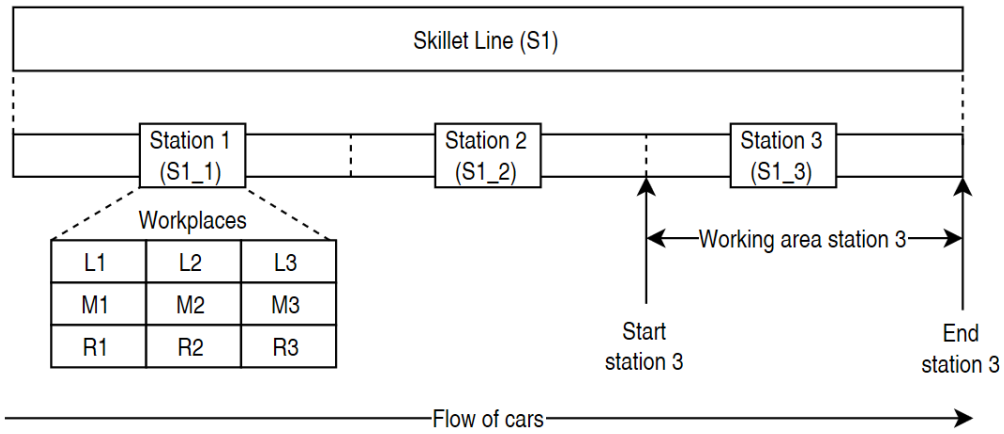


Figure 7. Lines, Stations and Workplaces

Frequency

Sometimes the best option is to let a single worker perform a job, instead of multiple workers. A reason for this can be that the working area is limited (a hard to reach location in or on the car) or the task at hand is indivisible. If this is the case, 'frequency' is used as a solution. This means that instead of having 3 workers (for example on workplaces L1 thru L3 in Figure 7) working on the car simultaneously, one worker moves with the car along the workplaces. More than one worker is put in a station, and each worker processes a car alternately. Worker number one processes car 1, and walks along with the car through the all the workplaces in the station. Worker number two works on car 2, and so forth. The station length might also be increased to give the worker enough time to finish the activities. If there are X workers doing their work as described above, the 'frequency' of the station is 1/X.

Double door line

As mentioned earlier, the double door line consists of two parallel lines. This is done because the processing time on most of its stations is approximately twice the takt. One line processes the even cars and the other processes the uneven cars. This gives the stations on each line double the time to complete their activities.

2.3 Workers and floaters

Each station in the assembly line has a minimum of one regular worker assigned to it. This worker has the required competences to work at this station (each station requires different competences). Workers are part of a team, which is assigned to a part of the assembly line.

Next to regular workers, floaters are used in the assembly line. These floaters are cross-trained workers that have the required competences for a part of the assembly line, and can work at any station in that zone. Most often these floaters are team leaders, but dedicated floaters are also used. Floaters can provide help at stations in case of a capacity overload, in order to reduce overtime. Currently there is no offline operational planning for the floaters. If an overload situation occurs in the line, a floater is requested at the station. If a floater is available at that point in time, it will move to the station to provide help. Floaters are assigned on a first come first serve basis.

2.4 Planning system

As mentioned in the introduction, a simulation based planning system has been developed by Ergo-Design for Company A. In this section we discuss the simulation model that is included in the system, why this model has been developed, what it contains, and how it works.

2.4.1 Goal of the planning system

The planning system has been developed to cope with the challenges that were discussed in Chapter 1. As mentioned earlier, line balancing is time consuming and cannot be performed on a daily basis, or for every change in demand. That is why an operational planning is needed to resolve remaining overload situations. Given a fixed allocation of tasks to stations, Company A performs production planning which determines the sequence of cars and the allocation of floaters. To support this process the simulation model was developed, with the goal of:

- Quantifying the effects of changing the sequence of cars that will be sent down the line, and constructing methods to construct efficient sequences (sequencing).
- Quantifying the effects of different allocation rules for floaters, and developing methods to schedule floaters in the assembly line in an optimal way (floater allocation).

We now discuss the contents and the functionality of the model.

2.4.2 Contents and Functionality

The current planning system is constructed in Tecnomatrix Plant Simulation. This software was developed by Siemens PLM Software and allows users to gain insight into their production processes by modelling them and simulating production runs. The model contains the final assembly as a whole and all lines and buffers that have been discussed in this chapter have been modelled. Workers are assigned to the stations and perform their activities as cars move over the line.

Production data is loaded into the model, which contains the cars that are in the storage tower. After it has been determined which cars will be produced and an initial sequence has been made (based on simple rules), the simulation is conducted. During the simulation relevant data are recorded that can be used for analysis. The starting/ending time of each car on each station is registered together with the processing time, used capacity, capacity overflow, and idle time per worker. It is possible to zoom in on parts of the assembly line for more details. Workers in the model change color according to the workload they are facing. Workers can change color ranging from blue (little workload) to dark red (high workload). This gives a visual indication of stations that have trouble keeping up with the takt.

After the simulation is completed, the processing times of the cars on each station can be used to optimize the sequence of cars that will be sent down the line (see 'Sequencing' section below). Based on this sequence floater allocation can be performed. The process of gathering information and performing optimization using the model is shown in Figure 8.

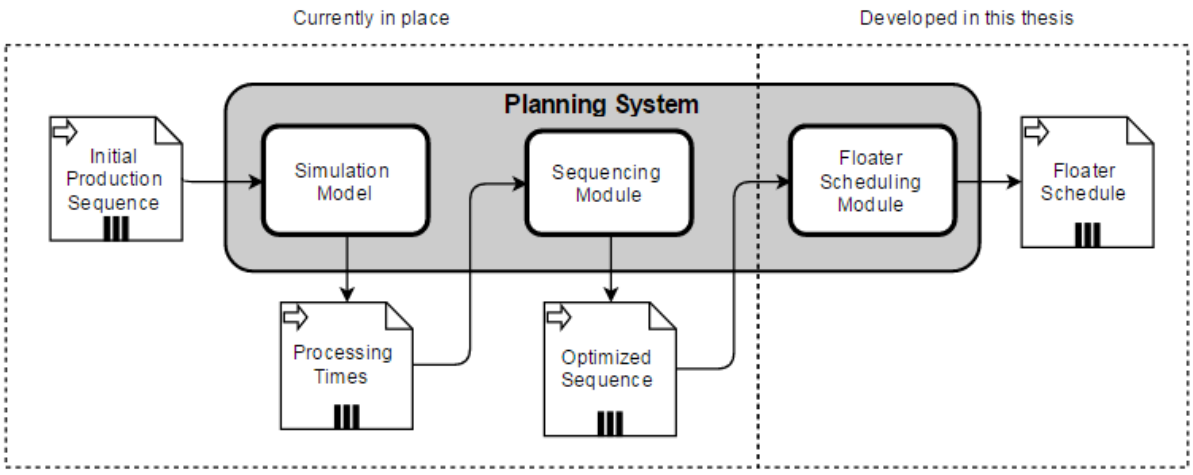


Figure 8. The planning system, its components, and information flow

Sequencing

The first goal of the simulation model is related to sequencing. In the old situation, sequencing for the assembly line was done purely based on mixing rules. Some examples of mixing rules are: only one in three cars can have a sunroof, only one in four cars of model x, and a maximum of two cars with air-conditioning in every four consecutive cars. Company A used 40 of such mixing rules. These rules have been developed to reduce overload situations in the assembly line, and are developed using a certain demand of cars. When this demand changes, the mixing rules become inefficient, resulting in (more) overload situations. Also mixing rules can result in a residual of difficult cars that remain unscheduled. Sequencing based on mixing rules had already been adapted into the simulation model in an automated way, in order to increase the speed of creating sequences and testing them by simulating the production run.

In the current planning system, a product sequencing module has already been developed (Op Den Kelder, 2015). Using the processing times that are produced by the simulation model, the sequence of cars was improved in order to reduce peaks in workload. To this extent, Op Den Kelder (2015) applied several heuristics such as Simulated Annealing to swap cars in the sequence and evaluate if this would decrease overload situations. In this way, a sequence of cars is constructed that minimizes overtime.

In this thesis, the sequence of cars to be sent down the line is considered as given. The optimized sequence that results from the methods of Op Den Kelder (2015) will be used as input in determining an optimal floater schedule/allocation.

Floater allocation

The second goal of the simulation model is to allocate floaters in an optimal way. As discussed earlier, there is no operational planning for the floaters yet, as they are assigned to stations while the assembly line is running by team managers. The simulation model provides possibilities to change this. Because the cars that have to be produced on a certain day are known in advance, overload situations in the line can be predicted given a sequence of cars and the allocation of tasks to the station (which is static).

With this information, it becomes possible to make a planning that makes the best use of floaters, e.g., each floater resolves as much overtime as possible. In this planning, travel distances and time can be taken into account to determine what overload situation can be reached from a certain station at a certain moment. If methods are developed for allocating these floaters in an optimal way, the company can then investigate what the effect is of increasing the number of available floaters for a production zone.

2.5 Conclusion

In this chapter the current situation in the final assembly of Company A has been described which provided insight into the configuration and properties of the final assembly. The current practices related to the allocation of floaters has been discussed.

The current method of floater allocation is done reactively without planning beforehand. If an overload situation occurs in the assembly line, the line manager assigns a floater (if one is available) to help resolve this issue. The allocation of floaters is done on a first come first serve basis, which may result in sub-optimal use of the floaters. More overtime can possibly be resolved if overload situations can be predicted and floaters can be scheduled accordingly.

Finally the goal, contents and the functionality of the simulation model have been described. This simulation model is used for operational planning, and will provide a platform for implementation for the solution design in this thesis.

3. Literature review

In this chapter, the existing literature that is relevant for this research is reviewed. In Section 3.1 we give an introduction to MMALs. In Section 3.2 and Section 3.3 literature is reviewed to determine the role of floaters in MMALs and what methods have been developed to schedule them. In Section 3.4 we discuss problems related to the problem of Floater Allocation in Mixed Model Assembly Lines (FAMMAL). In Section 3.5 we discuss and solution methods for the Vehicle Routing Problem (VRP). Finally, we discuss metaheuristics that are used to provide solutions to complex problems.

3.1 (Mixed Model) Assembly lines

The modern version of the assembly line was first utilized by Ransom Olds, who used it in the fabrication of a car called the Oldsmobile Curved Dash in 1901. Henry Ford and his engineers then improved many aspects of the assembly line such as installing a driven conveyor belt for the transportation of the products that were being assembled and standardizing the product. These improvements made Ford capable to produce a Model-T in 93 minutes in the year 1913. An assembly line that uses a means of transportation (most often a conveyor belt) to move parts from one station to another is called a paced assembly line. The pace of the conveyor belt is fixed, meaning that all activities at one station have to be completed in a certain amount of seconds called the 'takt'. Assembly lines (such as Ford's T-Model Line) on which a single standardized product is produced are called Single Model Assembly Lines (SMAL). Single model assembly lines are suitable for large-scale production due to low production costs, but in the modern world companies cannot always follow the high quantity low variability methods used by Ford. In the last decades, competition (not only in the automotive industry) has increased, and customers' needs and wants change quickly. These factors, together with increasing customer sophistication and internationalization, urge producers to offer an increased product variety in order to fulfill demand (Mac Duffie, Sethuraman, & Fisher, 1996).

In order to stay competitive, producers must find a way to increase product variety without a significant increase in costs. To do this, producers use line configurations that are adapted for the production of different models (Merengo, Nava, & Pozzetti, 1999). Multi-model Assembly Lines (MAL) can produce batches of different products, with (short) set-up times in between. In cases where more flexibility is needed, a Mixed-Model Assembly Line (MMAL) is used. MMALs are capable of producing multiple products with different configurations with negligible set-up times between batches and batch sizes as small as one (Merengo, Nava, & Pozzetti, 1999).

3.2 Floaters in MMAL literature

The major part of MMAL literature covers two main problems: the assembly line balancing problem and the (product) sequencing problem. In Table 2 we give an overview of the topics of the top 30 articles that were found using the search term "Mixed Model Assembly Lines" in Google Scholar and Science Direct.

Topic	Frequency of topic	
Product Sequencing	35 %	77%
Assembly Line Balancing	32 %	
Balancing & Sequencing	10 %	
Other (Mainly design and supply chain related)	23 %	23%

Table 2. Topics in MMAL literature

The fundamental line balancing problem deals with the problem of allocating task to the stations. Tasks are indivisible units of work. When allocating tasks, precedence relations have to be satisfied and some measure of effectiveness has to be optimized (Ghosh & Gagnon, 1989). Depending on line characteristics, optimization can be done with regards to multiple objectives. For paced lines, the rate of incomplete jobs is a candidate objective, whereas in un-paced lines the probability of blocking and starvation events becomes an objective. Other objectives include minimizing the number of stations in a line, and reducing WIP (Merengo, Nava, & Pozzetti, 1999).

The product sequencing problem deals with constructing a sequence in which products are to be sent down the assembly line. Again, multiple possible objectives can be optimized depending on line characteristics. Because processing times differ between products and stations in mixed-model assembly lines, a sequence has to be found that minimizes the rate of incomplete jobs, i.e. ensures that stations will have an acceptable workload. A simple example is to let 'difficult' cars (having long processing time) be followed by 'easy' cars (having short processing time). This is a basic variant of product sequencing based on a simple rule. Sophisticated techniques have been proposed by numerous authors, for example by Merengo, Nava, & Pozetti (1999) and Boysen, Kiel, & Scholl (2010).

In the articles discussing the two problems mentioned above, the usage of floaters is mentioned by the majority of the authors. The usage of floaters is indicated with statements such as 'floaters have to resolve overload situations' (Scholl, Kiel, & Scholl, 2010) or 'a floater is summoned by a regular worker in case of an overload situation' (Bock, Rosenberg, & Brackel, 2006). It becomes apparent that the usage of floaters in MMALs is a common phenomenon, but there has been little attention for the operational scheduling of these workers.

3.3 Floater allocation in the literature

Some researchers have attempted to fill the gap in the literature discussed in the previous section. In this section, we discuss papers that deal with the allocation of floaters in mixed model assembly lines.

Gronalt and Hartl (2000, 2003) develop a method to allocate floaters in a case study at a truck manufacturer. First, they calculate the overlap that each truck would cause on each station in the assembly line. Finishing work in the station downstream is called positive overlap, starting work in the preceding station is called negative overlap. When these overlaps o_{it} are calculated (for every takt t and station i) and a maximum overlap is defined, an infeasible situation might occur. Floaters (f_{it}) are then used to reduce overlaps and provide a feasible solution. The actual overlap O_{it} is then calculated as the original overlap minus the use of floaters ($O_{it} = o_{it} - f_{it}$). The minimum number of required floaters can then be calculated by finding the maximum number of floaters that is needed at any time in the assembly line. The authors generate the schedules for the floaters based on a heuristic that uses the shortest distance rule. The objective used is to minimize the travel distances of floaters, and it is assumed that all problematic situations have to be resolved by floaters.

Gujjala and Günther (2009) tried to decrease the number of required floaters by anticipative scheduling. Instead of using a floater when an overload situation occurs, the authors developed a method that can utilize floaters even before the overload situation actually takes place. The idea behind their method is that utility work does not have to be conducted at a fixed moment in time. For example, overtime at a station which results from a specific product can be resolved by finishing the preceding product earlier. The authors constructed intervals in which floaters can be used to resolve a problematic situation, called 'shiftable intervals'. Heuristics were then used to deploy floaters in these intervals. By using this anticipative method, the required number of floaters was reduced compared to the traditional approach.

The papers discussed above calculate overload situations and floater demand, based on information on processing times and product sequence. Mayrhofer, Marz and Sihn (2013) developed a simulation based planning tool to perform these calculations in larger and more complex cases. Using input data such as the sequence of products and task allocation, the tool calculates processing times and workforce requirements for every cycle at the production line. The model automatically requests a floater if one is needed, and this floater is assigned depending on availability. This gives planners the possibility of analyzing different scenarios, and plan the usage of floaters.

All models described so far have a deterministic nature. All processing times are fixed, and disturbances like the loss of a worker or a material bottleneck are left out of scope. To cope with these disturbances in the assembly

line Bock, Rosenberg and Brackel (2006) developed a method for real-time control of the production process. Every time a disturbance occurs in the assembly line, the model will adapt to the situation by quickly adjusting the production plan. This is done by generating a new initial ´solution´ to the production plan, and improving this solution using different heuristics that are executed in parallel. The employment of workforce and floaters is included in the tool that aims to minimize costs (consisting of repair, overtime and wage costs). In Table 3 an overview is given of the methods discussed in this chapter.

	Approach	Method	Floater availability	Goal	Floaters for all overloads?	Sample Size	
						Stations	Products
Gujjula & Günther (2009)	Planning	Exact	Unlimited	Min. # Workers	Yes	5- 10	n.a.*
Gronalt & Hartl (2000,2003)	Planning	Exact	Unlimited	Min. # Workers	Yes	16	20 x 44
Mayrhofer, März, & Sihn (2013)	Planning	Simulation	Unlimited	Forecasting	n.a.	n.a.	n.a.
Bock, Rosenberg, & Brackel (2006)	Real-time	Heuristic	Limited	Min. Total Costs	No	80-160	300-400

*Authors used a range of utility work occurrences during a 100 takt time span

Table 3. Overview of floater scheduling methods in literature

In the previous work regarding floater allocation some similarities can be found. Most papers assume that all overtime can be resolved by floaters. In real life situations where limited resources are available, the assumption of 'unlimited' availability of floaters is insufficient. Floaters have to be costly cross-trained to work at different stations and the availability of such personnel is most likely limited.

Another factor that is left out of consideration by the models used is the travel time of floaters. For example, Bock, Rosenberg, and Brackel (2006) state that in case of an overload situation, a floater 'moves immediately to the station'. Travel times can be a restricting factor for floater allocation, especially when the assembly increases in size. Floaters cannot be expected to be present at a station that is 150 meters away within a matter of seconds. For an operational planning to make sense, travel distances (times) have to be taken into account one way or the other.

Concluding, the literature gives insight into and some handles for the process of floater allocation, but does not provide an 'off the shelf' method for the problem at hand. We are dealing with a problem with characteristics and restrictions that are not yet discussed in the MMAL literature.

3.4 Related problems

In this section, we discuss problems that are well-known in the literature and share characteristics with the problem of Floater Assignment in Mixed Model Assembly Lines (FAMMAL). The main goal is to review if the methods that have been developed to solve them can be of use in solving the problem in this thesis. The problems that were identified as related problems are the Project Scheduling Problem and the Traveling Salesman/Vehicle Routing Problem, which we discuss in Section 3.4.1 and 3.4.2 respectively.

3.4.1 Project Scheduling

In Project Scheduling a project (consisting of a set of jobs) has to be completed. Each job has a duration and precedence constraints, and the sequence in which the jobs will be processed has to be chosen. In the *Resource Constrained Project Scheduling Problem* (RCPSP) the number of jobs that can be executed at a given moment is limited. The main goal of the RCPSP is to minimize the make span (duration) of the total project (Brucker et al., 1999).

The Recourse Constrained Multi Project Scheduling Problem (RCMPSP) considers the scheduling of multiple projects that demand the same resource. Kruger and Scholl (2009) introduce resource transfer aspects (for example moving personnel from one project to the other) in their model, and developed a heuristic based on job selection priority rules to solve it. A mapping of problem elements of FAMMAL onto Project Scheduling is given in Table 4.

FAMMAL	RCPSP	RCMPSP	Kruger and Schöll (2009)
Machines	-	Projects	Projects
Workload peaks	Jobs	Jobs	Jobs
Time of occurrence	Time Windows	Time Windows	-
Floaters	Resources	Resources	Resources
Travel time between stations	-	-	Sequence dependent transfer time

Table 4. Mapping of FAMMAL elements onto Project Scheduling elements

3.4.2 Vehicle Routing Problem

The Traveling Salesman Problem (TSP) describes the problem of a salesman that has to start his route from a fixed position (the depot), visit a number of cities exactly once, and then return to the depot. The objective of this problem is to minimize the total distance that is traveled. The Vehicle Routing problem is an extension of this problem, where a fleet of vehicles is used to serve a number of cities/customers. The vehicle routing problem incorporates capacity of trucks for deliveries, and has also been extended with time windows that prescribe when the vehicles are allowed to visit certain cities (Solomon, 1987). The objective of the VRP is to minimize the number of trucks used and the total travelled distance.

In the FAMMAL problem, one could view a floater as a unit that has to travel through the production line and 'visit' the workload peaks that occur. This description of the problem has similarities with the Traveling Salesman Problem and its extension the Vehicle Routing Problem, as shown in Table 5.

FAMMAL	TSP	VRP
Machines/Workload peaks	Cities	Cities
Time of occurrence	(Hard) Time Windows	(Hard) Time Windows
Floaters	-	Vehicles
Travel time	Travel time	Travel time

Table 5. Mapping of FAMMAL elements onto TSP/VRP elements

3.4.3 Review Related Problems

The objectives of FAMMAL and RCMPSP differ on a fundamental level. In RCMPSP terms, the FAMMAL problem tries to select and visit workload peaks (jobs) that reduce the overtime as much as possible. It is allowed to skip 'jobs', as not all overtime has to be resolved (a limited amount of floaters is available). In RCMPSP, all jobs have to be completed. The goal is to minimize the make span of the project, which is irrelevant to FAMMAL as the finishing time of the 'project' is fixed here (the duration of the production shift). Also, the jobs in FAMMAL can only be executed at a fixed moment in time (the time window in which the workload peak occurs). This is not a restriction to the project scheduling problem which is generally only restricted by precedence and capacity constraints. Multiple articles can be found that discuss the RCPSP with time windows (for example Brucker et al., 1999), but these time windows are created by setting a maximum duration for the project and deriving the earliest and latest completing time for each job. Such an approach is unsuitable for FAMMAL, as there can be 'gaps' in the floater schedules in which no workload peaks (jobs) have to be visited. These differences have lead us to discard the Project Scheduling approach for the problem in this thesis.

The TSP and VRP are also not suitable to be used as models for the FAMMAL problem. In FAMMAL, the sequence of workload peaks (cities) to be visited is not a decision variable. The objective function also differs fundamentally. The TSP aims to minimize the distance traveled, with the addition of minimizing the number of vehicles (floaters) for the VRP. As discussed earlier, the objective in FAMMAL is different from both these

objectives. And finally, the TSP and VRP both require all cities to be visited, which is also not a requirement to the FAMMAL problem.

Even though the TSP and VRP cannot be readily used as models for the FAMMAL problem, we believe that the routing aspect of the solution methods for these problems make them useful in this thesis for constructing initial solutions. Therefore, we discuss solution methods for the VRP next in Section 3.5.

3.5 Solution methods for the VRP

The TSP and VRP both have been proven to be NP-Hard (Solomon, 1978). This means that instances of these problems are hard to solve to optimality. Or as Blum and Roli (2003) state: "*complete solution methods [that guarantee an optimal solution] might need exponential computation time in the worst-case (…) this often leads to computation times too high for practical purposes".* Therefore, heuristics are applied to solve these problems. Heuristics are methods that do not guarantee an optimal solution to a problem, but provide a solution within reasonable computation time. The heuristic methods for solving the VRP can be divided into the following categories (Taillard et al., 1997):

- *Route construction heuristics*; construct routes sequentially (one by one) or in parallel (all vehicles at once) according to a set of rules.
- *Route improvement heuristics*;
- *Composite heuristics*; use both route construction and improvement methods.
- *Meta heuristics*; discussed in Section 3.6.

In Section 3.5.1 and 3.5.2 these heuristics are described in more detail.

3.5.1 Route Construction

Solomon (1978) proposed multiple heuristics for route construction for the VRPTW. *The savings method* starts by servicing each customer with an individual route. Then the savings that would result from adding a customer to another route are calculated. The customer pairs with the largest savings are merged into a single route consecutively.

The *time-oriented neighbor heuristic* is an expansion of the nearest neighbor method. Instead of only considering the closest distance, the time it will take before service can start at a customer is also taken into account, together with the urgency of service. Values for each possible customer that can be added to a route are calculated as a weighted sum of distance, time and urgency, and the largest value is added to the route. A new route is started if there is no customer that can be added to the end of the current route.

The *insertion* heuristic starts by selecting either the furthest non-routed customer or the customer with the earliest deadline. At each iteration a customer is then selected according to two criteria c1 and c2. For each non-routed customer its best insertion place in the route is determined using c1. Then c2 calculates the best customer to add to the route. If no more customers can be added, a new route is initiated. Different forms of c1 and c2 are examined.

The final heuristic discussed by Solomon (1978) is a time-oriented sweep heuristic. It uses a clustering and a tour building heuristic repeatedly. The customers are clustered using the traditional sweep heuristic, placing customers with similar angles with relation to the depot in the same cluster. Then a tour building heuristic is used (insertion in this case) to build tours for each cluster.

3.5.2 Route Improvement

Route improvement heuristics aim to improve the routes of the vehicles in the VRP by iteratively changing small parts of routes. For example, the classical k-opt exchange method selects k links between customers, and replaces them with k links (Potvin & Rosseau, 1995). Or-opt heuristics select a sequence of one, two or three

customers and tries to insert it somewhere else in the route (Potvin & Rosseau, 1995). These heuristics stem from classical TSPs, and many variants have been developed, to accommodate for different problem types such as the VRP (with time windows). The 2-opt* procedure exchanges two links from different routes such that the orientation of the route is preserved (Potvin & Rosseau, 1995), which is a desirable feature for problems with time windows. Route improvement heuristics are often used in the framework of meta-heuristics (such as those discussed Section 3.6) to create neighbor solutions.

3.6 Meta Heuristics

Optimization problems in which the objective function is defined on a finite domain (e.g. there is a finite number of choices that can be made for the decision variables), are called *combinatorial optimization problems*. The solution to this sort of problems might seem straightforward at first: list all the possibilities and choose the best solution; this is called (complete) enumeration. But for most practical problems, the total number of possible solution becomes so large (even for moderate problem sizes) that listing all of them becomes impossible (Graham, 1995). When the objective function is too complicated, and/or the problem size is too large, it is often impossible to find an optimal solution. In these cases, approximate algorithms called heuristics can be developed that produce reasonably good solutions (Graham, 1995).

Algorithms that combine basic heuristics (such as those discussed in Section 3.5.1 and Section 3.5.2) in a higher level framework aimed at efficiently and effectively searching a solution space, are called *metaheuristics*. Metaheuristics are often used to find solutions to combinatorial optimization problems. Some properties of metaheuristics include (Blum & Roli, 2003):

- Metaheuristics are strategies that 'guide' the search process.
- Metaheuristics may incorporate mechanisms to avoid getting trapped in local optima.
- Metaheuristics are not problem specific.
- Metaheuristics are approximate and non-deterministic.

Therefore we discuss three different types of metaheuristics: Simulated Annealing (SA), Tabu Search (TS) and Genetic Algorithms (GAs). These three heuristics have been selected as they have been studied extensively and their inner workings differ substantially. For example, SA and TS use a single solution based approach whereas GAs use a population based approach.

3.6.1 Simulated annealing

Simulated Annealing (SA) is a heuristic that can be used to obtain solutions to large and complex optimization problems. The process in which it finds a solution is very similar to the process of annealing solids. SA was created when Kirkpatrick, Gelatt, and Vecchi (1983) adapted a model that describes the cooling of a solid, to guide the search process in a large and complex solution space.

The following is a description of SA applied to a minimization problem. Given a finite solution space, every solution in this space has an associated cost determined by a cost function. An initial solution is chosen, sometimes at random. By making (small) adjustments to this solution, a so called neighbor solution is created. This neighbor solution will result in a change of the cost function ð. If we create several neighbor solutions, and accept the one with the best change in cost function ð, we are performing what is called a descent algorithm. One downside of this method is that it gets stuck in local optima. This occurs if in order to reach the global optimum, we must first move to a solution that has a worse cost function than the current solution.

Simulated Annealing tries to avoid getting trapped in local optima by sometimes accepting a neighborhood move that increases the cost function (in the case of minimization problems) (Eglese, 1990). The probability of accepting a move that deteriorates the objective function with a cost of ∂ is determined by a function of the form $e^{(-\partial/\mathrm{T})}$. The temperature T is a value that decreases as the algorithm proceeds, so the probability of

accepting a worse solution also decreases. The function also implies that small increases ð are accepted more often than large increases. The parameter T is chosen such that initially virtually all moves are accepted, e.g. ratio of proposed to accepted moves (acceptance ratio) is approximately equal to 1. The temperature is then decreased according to a cooling schedule during the search process. In this way the search process accepts bad moves often at the start, but slowly solidifies around an optimum.

The complete process, works as follows. After choosing an initial solution and setting the initial temperature, multiple neighbor solutions are created according to a neighborhood structure. Each neighbor solution is accepted if its cost function is at least as good as the current solution, and is accepted with probability $e^{(-\partial/T)}$ if it results in an increase of cost. After a predetermined number of neighbors *N* have been considered, the temperature is lowered according to a cooling schedule. Again, a number of *N* neighborhood solutions are considered. This process continues until a stop criterion is met, for which multiple options are available. Examples of stopping criteria are: a predetermined running time has elapsed; a number of iterations (temperature changes) has been completed; a set stopping temperature has been reached; or a set number of iterations has gone by without an improvement in the best found solution.

3.6.2 Tabu Search

Another widely used optimization heuristic is called Tabu Search (TS). The process starts with an initial solution and iteratively moves to another solution by performing a move. The set of solutions that can be reached from a solution is called the neighborhood. In each iteration, the best neighbor solution is selected and this move is stored on a Tabu list. TS forbids or penalizes moves that are on the Tabu list, guiding the search process away from recently visited states (Glover, 1989). The Tabu list has a length T, and when T moves are already stored, the 'oldest' move is removed from the Tabu list. The algorithm will forbid a move on the Tabu list, unless it fulfills some sort of aspiration criterion, for example if the move will result in the best solution so far. This enables the exploration of new promising solutions that can only be reached by performing a move from the Tabu list. This is desirable as in many cases a solution is not only defined by the moves that led to that solution, but also the sequence in which they are performed.

In addition to the short term memory (Tabu list), Tabu Search can be extended with more long term-oriented properties such as intensification and diversification schemes. Intensification will focus the search on a region of solutions that contain elements which were consistently present in good solutions. Diversification schemes will guide the search process to other regions, and alternating between diversification and intensification will provide the most effective search (Glover, 1989).

3.6.3 Genetic Algorithm

Genetic Algorithms (GAs) are metaheuristics that can be used to solve optimization problems. The power of GAs lies in the fact that they are robust and can handle a wide range of problems successfully. Some applications of GAs include combinatorial optimization, machine learning and design problems (Beasley, Bull, & Martin, 1993). GAs use the principles of evolution to 'evolve' solutions towards optimal ones. We now give a global description of the basic GA process, which is shown in Figure 9.

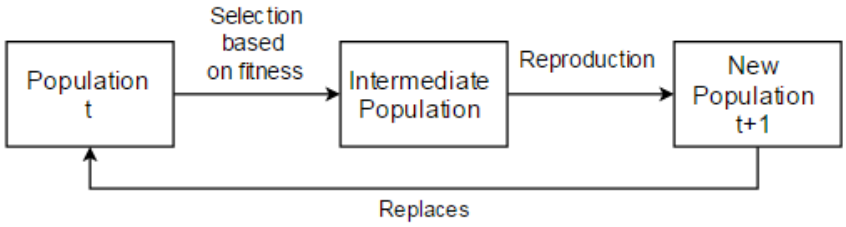


Figure 9. Basic Genetic Algorithm process

GAs work with a population, which is a collection of solutions. The highly fit members of the population (individuals) are then put in an intermediate population. Individuals from this intermediate population are then given the opportunity to reproduce by cross breeding with other individuals. This creates a new generation of offspring, which then (partially) replaces the previous population. Because highly fit individuals have a higher chance of reproducing, their characteristics will be more prevalent in the next generation.

Encoding and fitness function

Typically there are two elements of GAs that are problem dependent, the problem encoding and the evaluation function (or fitness function) (Whitley, 1994). Before GAs can be executed, a suitable problem encoding has to be formulated. For many problems, a solution can be formulated as a set of parameters. These parameters, known as 'genes', then form a string of values called a 'chromosome' (Beasley, Bull, & Martin, 1993). In a design example, we might want to design a bridge that has the best strength to weight ratio. The string of values could then represent the dimensions of the beams of the bridge.

Before we can evaluate the fitness of a chromosome, it has to be translated to the individual that it represents. In the bridge example, the dimensions of the bridge have to be translated to a complete bridge design before we can determine its strength to weight ratio. The fitness function is used to perform this translation, and then determine the fitness/quality of the individual. The fitness function differs for each problem, and should measure the quantity we want to optimize. In the bridge example, a bridge with a large strength to weight ratio should be assigned a high fitness value.

Initial population

A GA has to be initialized with an initial population. This population can be generated by creating random solutions/individuals, to ensure a wide coverage of the search space. Sometimes heuristic methods are used, which will 'seed' the GA in areas of the search space where optimal solutions are likely to be found (Beasley, Bull, & Martin, 1993).

Selection

From each generation, some individuals are selected for reproduction. This selection is performed based on the relative fitness of an individual compared to the rest of the population, favoring those individuals with better fitness scores. The selection process consists of two steps. First the relative fitness of each individual is converted into a chance to be selected (distribution). Next, individuals are selected by sampling this distribution. Four commonly found selection schemes are proportionate reproduction, ranking selection, tournament selection and Genitor (or steady state) selection (Goldberg & Deb, 1991).

In *proportionate reproduction*, the probability p_i of an individual i to be selected to produce offspring, is proportionate to its fitness value f_i compared to the average fitness value of the generation, or in formula form: $p_i = f_i / (m * \overline{f_i})$ (Goldberg & Deb, 1991). Where m is the number of individuals in the population. Proportionate reproduction has the negative side effect that when there are a few 'super fit' individuals in the population, their characteristics will quickly start to dominate the population leading to premature convergence.

Fitness ranking is a selection scheme that prevents this from happening. The population is sorted on their fitness value in non-increasing order. The rank of each individual is then used to calculate the probability of being selected, with a lower rank having a lower probability of being selected. The relation between rank and selection probability can be linear (linear ranking) or exponential (exponential ranking). Because probabilities are based on rank and not fitness values, the influence of extreme individuals is negligible. Fitness ranking has been shown to be superior to other methods of dealing with extreme individuals, such as *fitness scaling* (Beasley, Bull, & Martin, 1993).

Tournament selection compares *n* random individuals from the population, and selects the best one for reproduction. *Genitor selection* works by selecting one individual based on linear ranking, which replaces the worst individual in the population. Goldberg and Deb (1991) they found that adjustment of parameters can make all selection methods discussed above (excluding proportionate reproduction) show similar performance.

Steady state vs generational replacement

Genitor selection differs from other selection schemes because it replaces only one (worst) individual of the population at a time, which is called steady state replacement. Other selection methods apply generational replacement, in which the whole population is replaced in each generation. Goldberg and Deb (1991) found no evidence that steady state replacement performs better than generational replacement.

Elitist approach

An elitist approach in GAs is an approach that considers both the parents and newly created offspring simultaneously, and selects the best individuals. Steady state replacement discussed above is one example. Another GA algorithm that uses an elitist approach is the NSGA-II algorithm proposed by Deb et al. (2002). NSGA-II has received a lot of attention in the literature and has shown good performance. The NSGA-II algorithm combines parents and offspring, and selects the fittest 50% of these individuals as the population for the next generation. This process is shown in Figure 10, which shows the differences with the classical approach.

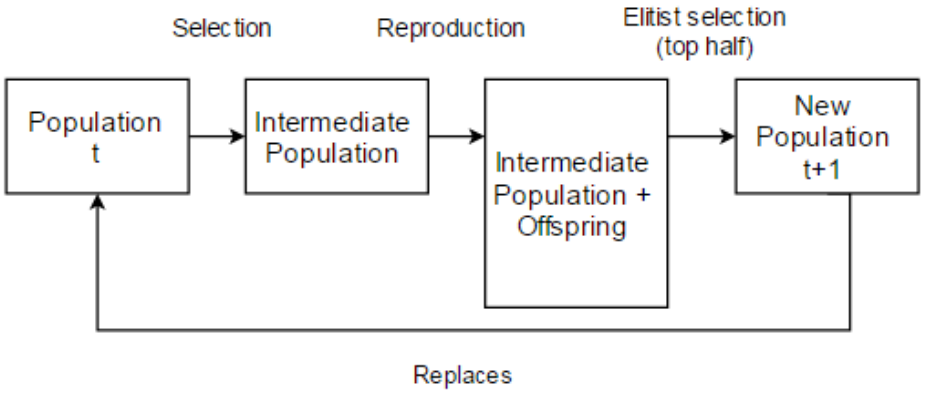


Figure 10. NSGA-II process

Reproduction

When two individuals are selected to produce offspring, a crossover operator is performed on their chromosomes with a probability $P_{crossover}$. The value of $P_{crossover}$ is typically chosen between 0.6 and 1 (Beasley, Bull, & Martin, 1993). A crossover is performed by choosing a random position on the chromosome string. At this position the chromosomes of the two parents are split, and the rear portion is swapped between parents, producing two children (see Figure 11). If no crossover is performed, two duplicates of the parents are placed in the next generation.

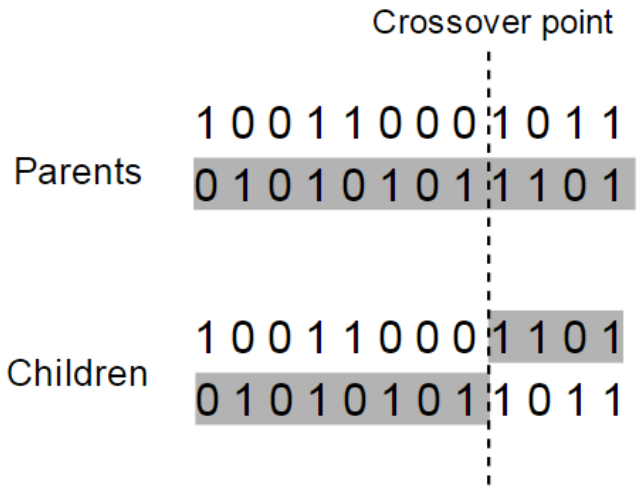


Figure 11. Crossover operator in Genetic Algorithms

To finalize the reproduction process, mutation is applied to all children. Each gene in their chromosomes has a small probability of being altered. In Figure 12, a mutation is performed on the 6th gene of the chromosome.

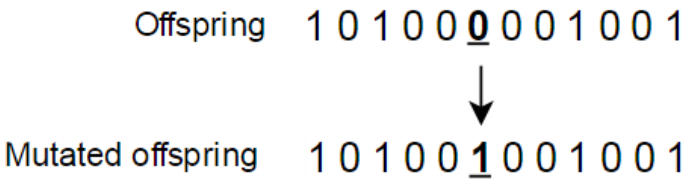


Figure 12. Mutation operator in Genetic Algorithms

The chance of any gene being mutated (mutation rate) is usually kept very low, and often smaller than 1%. Amongst others, Bäck (1993) found 1/n to be the best performing mutation rate on various problems, where *n* is the number of variables (genes) per individual.

3.7 Conclusion

In this chapter, we created a theoretical framework for our problem by reviewing MMAL and optimization literature. Floaters are widely used in MMALs, but there is a lack of attention for the operational scheduling of these floaters in the literature. Some authors have tried to fill this gap, but none of the methods discussed in the literature was deemed suitable for the problem at hand. Two related problems were identified, the Project Scheduling Problem and the Traveling Salesman/Vehicle Routing Problem. The models found in the literature do not provide an off-the-shelf solution to our problem, but constructive heuristics for the VRP have been identified as useful for creating initial solutions due to their routing aspect, and have therefore been discussed. Finally, we reviewed metaheuristics (Simulated Annealing, Tabu Search and Genetic Algorithms) that are used to provide solutions to combinatorial optimization problems.

4. Solution Design

In this chapter, we propose a solution method to construct floater schedules aimed at minimizing overtime in the assembly line. We first identify the functional requirements to the solution method in Section 4.1. In Section 4.2 we give a detailed problem description from which we formulate a model in Section 4.3. In Section 4.4 we make choices regarding our solution approach, and in Section 4.5 we propose constructive heuristics to create initial solutions. Finally, we propose optimization methods in Section 4.6 to improve the initial solution.

4.1 Functional requirements

The solution method discussed in this chapter is implemented into the Planning System discussed in Chapter 1 as an additional module. Functional requirements prescribe what the tool should be able to do, and thus should be taken into account during the design of the solution method.

Together with Ergo-Design B.V., the following functional requirements have been formulated:

1. Output: given a fixed number of floaters, the solution method should produce one schedule per floater, showing where he should provide assistance at which moments in time.
2. Running time: the tool should have a running time that does not exceed 30 minutes, keeping in mind that schedules have to be created every day.
3. Ease of use: The tool should create floater schedules with the click of a button.

Another desirable property of the solution has been identified as:

4. Flexibility: the tool should be adaptable to other situations, keeping in mind the possible deployment at other facilities in the future.

4.2 Detailed problem description

In this section, we give a detailed description of the problem, describe the configuration of working stations in detail, and give more insight into how floaters resolve overtime in the assembly line. We also provide formulas that describe the problem and its restrictions in more detail.

We describe the current situation in the final assembly of Company A in Chapter 2. In the current situation, floaters are assigned to stations on which a capacity overload situation (overtime) occurs, based on the First Come First Serve principle, if one is available. It became clear that an (off-line) planning for floaters is missing, and that filling this gap could result in improvement of the performance of the assembly line, i.e., less rework costs.

The problem at hand can be stated as follows: given a planning horizon of one shift (8 hours) and a sequence of cars to be produced during that shift, a schedule has to be constructed that tells specifies when and where (which station) a floater should provide support. The goal is to optimally use floaters to minimize overtime in the final assembly.

4.2.1 Stations

Each station has a working area, and cars pas through that area in the duration of one takt (52.8 seconds). Workers move along the line together with the car while working on it. Additionally, workers can move out of their own station for a limited distance called overlap. Stations that have this characteristic are called ‘open stations’. Stations are not closed cells, but marked working areas alongside the assembly line in which tools and workers are placed. Workers can start their activities earlier by moving into the preceding station, or workers can keep on working on the car for a longer amount of time by moving into the following station alongside the car (see ‘Earliest start’ and ‘Latest Finish’ in Figure 13). The overlap at the beginning and the end of the station are called negative and positive overlap respectively. If the workers have not completed their activities when the car reaches the end of the positive overlap of the station, they are forced to leave the work unfinished and

move back to start processing the next car. The amount of seconds that would have been needed to finish the car is called 'overtime'.

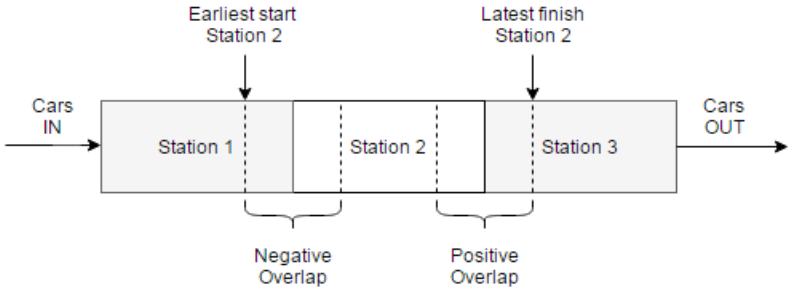


Figure 13. Station Details

4.2.2 (Non-) Conflicting Overlaps

The overlaps discussed in Section 4.2.1 come in two forms. If the assembly line is configured in a way that workers from a certain station cannot perform their activities while the previous station is still working on the car, we are dealing with conflicting overlaps. If the overlaps in the example of Figure 13 were of the conflicting type, station 3 would have to start its activities on a car later if station 2 was still working on the car in the positive overlap area. And thus, overtime at a station would also propagate to other succeeding stations.

We assume that stations in the assembly line can work simultaneously on the same car in the overlap area. This property is called non-conflicting overlaps in the literature, and limits the consequences of overtime to the station on which it originated. As a result, non-conflicting overlaps reduce the complexity of the overtime calculations compared to conflicting overlaps (see Section 4.3.3). It has been indicated by Company A that non-conflicting overlaps resemble the real-life situation in their assembly line.

4.2.3 Overtime

When workers cannot finish their activities within the limits of the station, we are dealing with overtime. Overtime on a station can be caused by two factors, or a combination of the two:

- The processing time of a car is longer than the takt.
- The workers start their activities (too) late, because they needed extra time on the previous car.

We now illustrate how overtime is caused, and how the position of the workers in a station changes over time (see Figure 14).

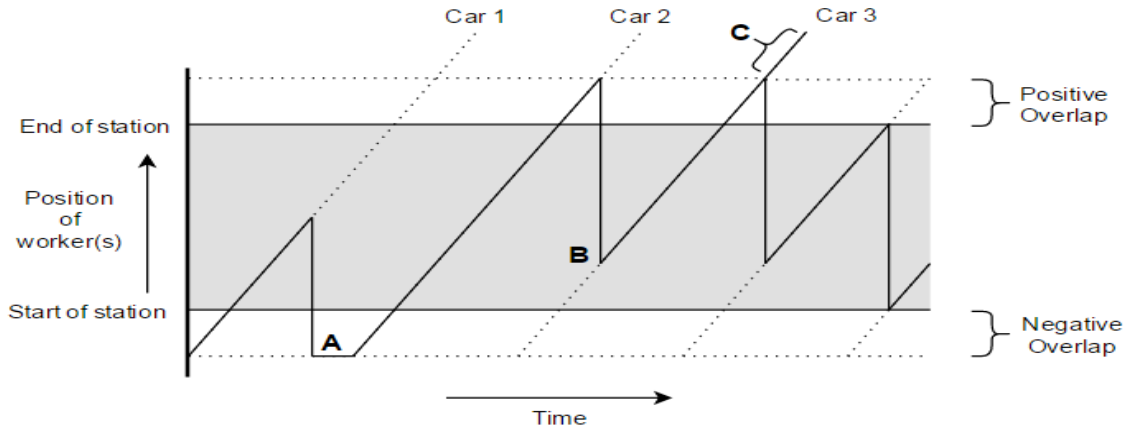


Figure 14. Development of overtime

In Figure 14, the location of the worker(s) and time are displayed on the y- and x-axis respectively. At the start of every takt, a car enters the station, indicated by the diagonal dotted lines. These are the 'paths' the cars follow. Car 1 enters the station first, having a processing time smaller than the takt. The worker performs his activities on the car and then returns to the start of the station and waits for Car 2. The idle time is indicated with 'A' in Figure 14. Car 2 has a processing time longer than one takt, and the worker is finished the moment the car reaches the end of the positive overlap. He then moves back to the next car, and starts processing it at 'B'. Notice that this is not the earliest starting time. As a result, Car 3 cannot be finished before reaching the end of the positive overlap, resulting in overtime indicated with 'C' in Figure 14.

4.2.4 Resolving overtime with floaters

To resolve the overtime that occurs in the assembly line, floaters are used. These cross-trained workers are deployed in the assembly line to resolve overtime by assisting regular workers at different stations. The function of floater is often occupied by the team leader of a part of the assembly line, but designated floaters are also used. It is assumed that floaters have the same productivity as regular workers, i.e., they can perform 52.8 seconds of processing time in one takt.

To illustrate how floaters resolve overtime, we give the following example. In Figure 14 we gave an example of overtime as a result from a car with a long processing time. Suppose we want to resolve the overtime on Car 3 (indicated with 'C' in Figure 14). In Figure 15 a floater is deployed to perform work on Car 3.

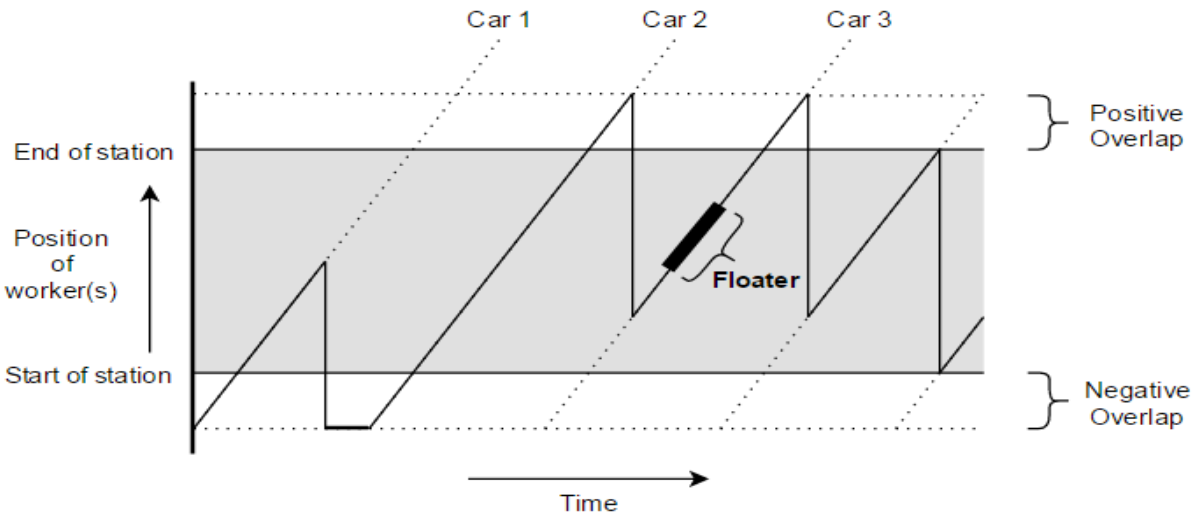


Figure 15. Effect of floater deployment

The bold line indicates that a floater is providing assistance at the station. The amount of work that has to be done by regular workers is decreased, with the result that the overtime from the previous example has been resolved, i.e., the workers can finish their activities within the bounds of the station. We can also choose to let a floater provide help at a station in anticipation of overtime. For example, by letting a floater assist on the car preceding the problematic car, the station will build up a 'lead' so the station can start earlier on the next car.

To illustrate the multiple ways in which overtime can be resolved, we discuss a theoretical example below. We use the same method of visualization as in Figures 15. In Figure 16 the original situation is shown, in which overtime occurs on Car 3 and 4. Suppose we want to resolve the overtime that occurs on Car 3. We have multiple options to do so. In Figures 17-19, we have illustrated the effects of deploying floaters in three different ways. Again, the bold lines indicate the deployment of a floater.

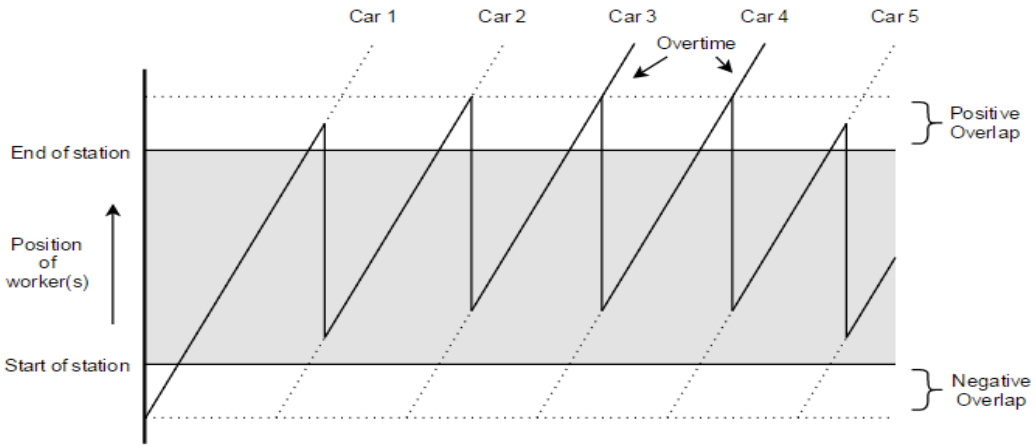


Figure 16. Station with overtime, original situation.

In Figure 17, the overtime is resolved by employing a floater on Car 3. The duration of stay of the floater is equal to the amount of overtime. This resolves the overtime on Car 3 but has no effect on the following car. In the original situation, the workers had to move back because the car reached the end of the overlap. In Figure 16, the workers still have to work until the car reaches the end of the overlap (with the difference that the activities on the car are completed).

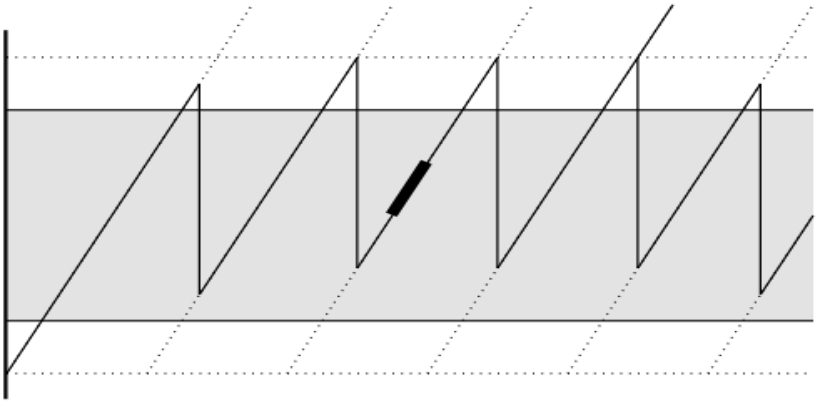


Figure 17. Assistance on Car 3, overtime on Car 3 resolved.

In Figure 18, a floater is again deployed on Car 3. This time the duration of stay is chosen such that the overtime is resolved with the addition that the floater stays for some extra time, such that the regular workers can move back to the next car the moment they reach the end of the regular station bounds. Now workers can start activities on Car 4 earlier relative to the original situation (Figure 16), as they can move back to the start of the station earlier to start working on it. As a result, the overtime on Car 4 is also resolved.

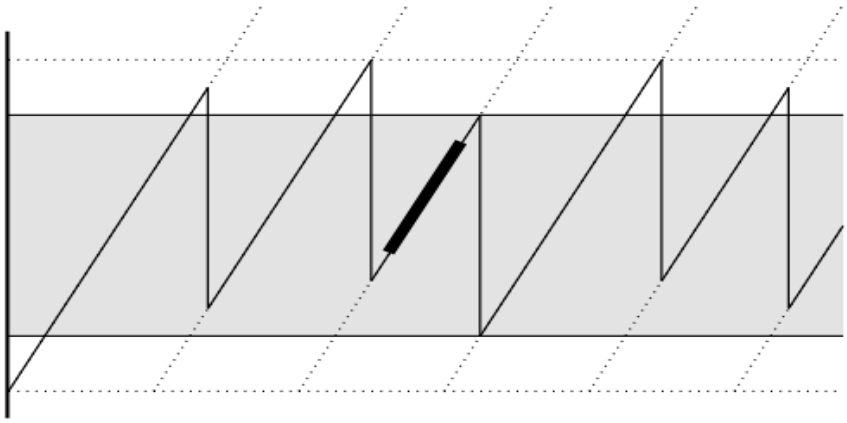


Figure 18. Assistance on Car 3, overtime Car 3 resolved, earlier start on Car 4, overtime Car 4 resolved.

In Figure 19, a floater is assigned to Car 2. In the original situation (Figure 16), the workers finish their activities on this car in time (no overtime on this particular car), but the start of activities on the next car is delayed with regards to the earliest starting time. Floater assistance on Car 2 thus reduces the starting time on Car 3. The result is that the overtime on Car 3 is resolved. However, Car 4 still has overtime.

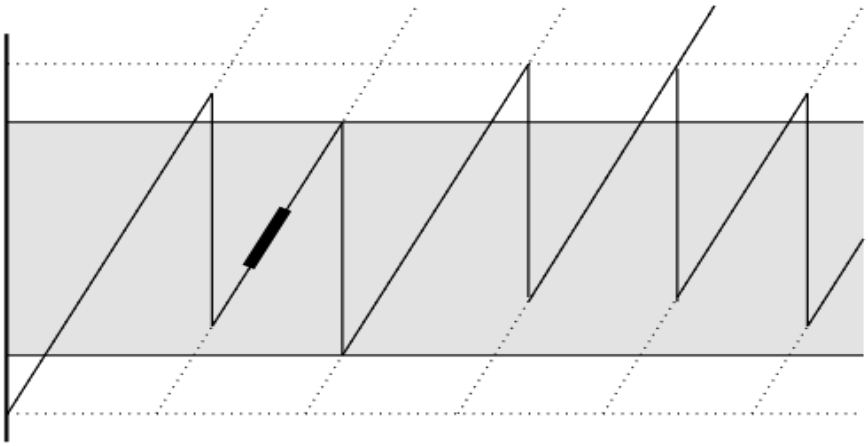


Figure 19. Assistance on Car 2, earlier start Car 3, overtime Car 3 resolved.

4.2.5 Properties and Assumptions

The following list summarizes the main properties of interest of the assembly line:

1. Cars are transported on a constant-speed conveyor-belt. Also, the sequence of cars does not change once the cars have entered the final assembly (no use of re-sequencing).
2. The takt time of the assembly line is fixed, and the cars are placed on the conveyor at a constant interval.
3. Workers move downstream along the assembly line while working on a car. When they finish working on a car, they will move back (towards the start of the station) to the next car in the sequence.
4. Workers can only work within the limits of their station, plus a predefined overlap before and after it. These overlaps (see Figure 13) specify how far a worker can move into the previous station (to start earlier) or into the next station (to keep working on a car). This distance is limited due to the availability of power tools and other factors. For experimental purposes, in this thesis overlaps do not differ between stations and are set at 5 seconds (roughly 10% of the takt time) for both negative and positive overlap.

Additionally, we made some assumptions to translate the real life situation into the model for which we develop a solution method. These assumptions have been discussed with Company A and Ergo-Design B.V., and are deemed reasonable for this application by all parties involved. In the list below we summarize these assumptions:

1. The processing time of a car at a station is deterministic.
2. The traveling time within a station from one car to the following car is negligible.
3. There is no interference between floaters and regular workers.
4. Floaters have the same productivity as regular workers.
5. Only one floater can provide help at a station at the same time.

In Chapter 6 we discuss how some of these assumptions influence the solutions that are obtained.

4.2.6 Car/Station Combinations (CSCs)

So far we have considered examples for one station. The assembly line consists of multiple stations, through which cars pass during production runs. For the remainder of this thesis, we refer to a car being at a certain station as *Car/Station Combination, or CSC*. Calculations of quantities such as overtime are based on CSCs. For example, if Car 3 causes 20 seconds of overtime on Station 11, the CSC (Car 3/Station 11) has 20 seconds of overtime. Every single CSC is also numbered.

4.2.7 Time windows

The entry and exit time of a car on a station (CSC), with one takt (52.8 seconds) between the two times, form a time window in which work can be done on the car (for the regular station bounds). In this thesis, this window is extended with the negative overlap at the start, and the positive overlap at the end. This is done because non-conflicting overlaps are used, which means the workers are free to work on the car in the overlap areas without hindering other stations. The regular workers and a possible floater can start working on a car the moment it reaches the negative overlap of the station. Thus, we note the time window associated with a CSC in which the car is available for work as [Entry, Exit].

At Company A, all CSCs have a processing time smaller than two times the takt at each station given the number of workers at each station. Thus, overtime at a single CSC will never exceed a single takt worth of processing time. The total time between the entry and exit time of a CSC is one takt plus the negative and positive overlap. This means that overtime at any CSC can be resolved by a single floater if he arrives early enough.

We conclude by noting that the characteristics described above are applicable to other assembly lines in most cases, as processing times longer than two times the takt would consistently cause problems in any assembly line. Such assembly lines would therefore have to be rebalanced (changing the allocation of tasks between stations).

4.2.8 Travel and set-up times

The task ahead is to determine schedules for each available floater, prescribing where a floater has to be at what specific moment. If a floater moves from one station to the other, travel times should be taken into account. The travel time between two stations depends on the distance d_{ij} between the starting station i and target station j and the walking speed of floaters ($travel_{ij} = d_{ij} / floaterspeed$). For experimental purposes, we assume in this thesis that floaters travel with a normal walking speed of 5 km/h (1,39 m/s).

The second factor that plays a role in moving from one station to the other is setup time. It is logical to assume that if a floater were to travel from one station to another, some setup time is needed before he can start his activities. The floater might have to grab the tools he needs, position himself within the station and receive instructions in one form or the other. We assume a constant setup time on each station, called s_j.

The total time needed for a floater to switch to another station t_{ij} is the sum of the travel time and setup time, or in formula form:

$$t_{ij} = \begin{cases} travel_{ij} + s_j & if \quad i \neq j \\ 0 & otherwise \end{cases}$$

If a floater stays at the same station to provide help on the next car, this time is negligible, i.e., $t_{ii} = 0$.

4.3 Model

Based on the problem description discussed in Section 4.2, we now construct a model for which we propose a solution method later on.

4.3.1 Decision Variables

The solution method developed in this thesis produces schedules for the floaters. A Car/Station Combination (CSC) 'occurs' in a fixed time window (as discussed in Section 4.2.7), because the assembly line moves at a fixed rate, launch rates are constant and no resequencing is done after the cars have been put on the assembly line. This means that a list of CSCs to be visited by a floater describes the sequence in which they will be visited, and thus the 'route' of the floater is known.

For each CSC we have to decide by which floater it will be visited (if any). This is the first set of decision variables. The second set of decision variables indicates a duration of stay in seconds for each CSC visited by a floater. Thus, a complete solution consists of the following two sets of decision variables:

- A list of CSCs to be visited by each floater.
- The duration of stay of each floater at each CSC in seconds.

To visualize the floater schedules and how they are constructed and optimized, we use the representation shown in Figure 20.

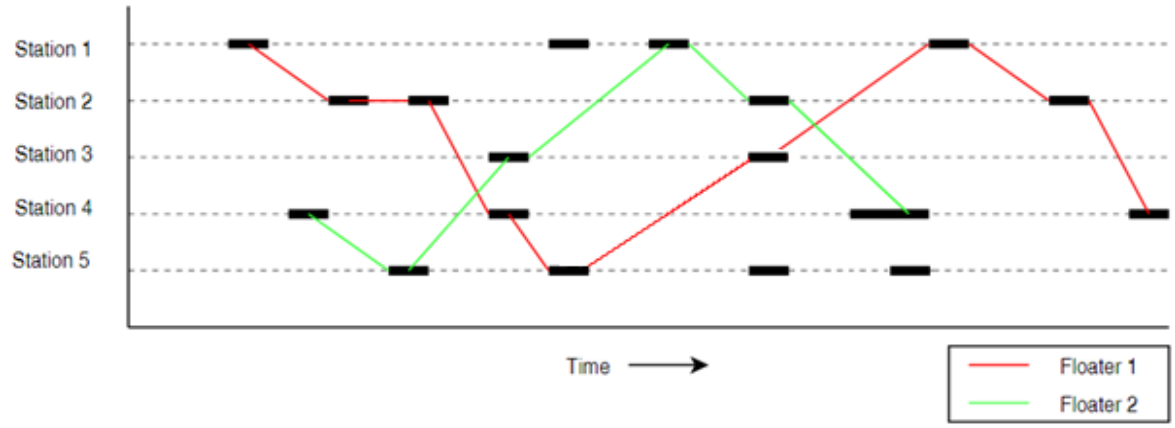


Figure 20. Graphical representation of floater schedules

Stations are shown on the y-axis, with 5 stations indicated by the dotted horizontal lines. Time is displayed on the x-axis. The bold lines show the time windows (the window [Entry,Exit] as discussed in Section 4.2.7) of CSCs that have overtime and/or are visited by a floater. Floater schedules are indicated by the colored lines (representing each floater) that move along the CSCs and towards the right. Floaters do not have to start the moment the car enters the station, and can also leave earlier. This is shown by the colored floater lines touching the bold lines (time windows) at a position other than at the far left or right.

4.3.2 Objective Function

The goal of our solution is to optimally use floaters in reducing overtime and the resulting problems in the final assembly. Overtime is the workload that is left on the car when it leaves the positive overlap of the station; these cars are unfinished and have to be repaired at a touch up station or at the end of the line.

In determining the objective function of our problem, we have to determine what factors indicate the quality of a solution. Together with Ergo-Design B.V. we identified the following candidates:

1. The total amount of overtime on all cars
2. Total number of defects (number of overtime occurrences)
3. Percentage of First Time Right cars, %FTR (percentage of cars without defects on any station)

After a solution has been generated, the first two quantities can easily be calculated (see Section 4.3.4). Candidate three has been discarded, as Company A has indicated that the main objective is to minimize the overtime and the number of defects on the cars. Therefore we chose to use factors 1 and 2 in our objective function during optimization, while also reviewing the performance of the different solution methods regarding factor 3.

The final goal is to reduce the costs of rework due to the defects made on cars. For every defect on the car, we assume a setup time is needed for it to be repaired. Tools and/or parts have to be gathered, and possibly some parts have to be removed in order to access the place where the defect has to be repaired. Therefore, we prefer a solution that has one defect of 30 seconds of overtime, over a solution with three times 10 seconds of overtime. A consideration has to be made how much weight we give the total amount of overtime (∑OT) and the total number of defects (∑D) in the objective value. In formula form:

$$Objective\ Function = w1 * \sum OT + w2 * \sum D$$

The weights w1 and w2 depend on practical issues that determine the costs of both quantities they relate to. As these costs are not exactly known, we keep them variable in the final model. To obtain results, we have chosen w1 = 1 and w2 = 10. This means that one defect equals 10 seconds of overtime in terms of costs. Two

cases of 10 seconds of overtime ($\sum OT = 20$, D=2,) results in an objective function of 40, whereas one case of 20 seconds of overtime ($\sum OT$ =20, D=1,) has an objective value of 30. An analysis of the influence of these weights can be found in Section 5.10.

The goal of the floater scheduling process is to minimize the objective function, given a limited amount of floaters (specified per area, see Section 4.4.1).

4.3.3 Calculations and restrictions

In this section, we provide a basis for the calculation of all necessary quantities. We distinguish between two types of calculations. The first type involves station calculations. To perform these calculations, the following inputs are required:

- Processing time of each CSC (output of the simulation model, see Section 2.4).
- The amount of work performed by floaters at each CSC (decision variable).
- CSC characteristics (fixed parameters such as entry time, exit time, overlap sizes, etc.).

These calculations provide us with the overtime on each CSC. This information is used to evaluate the quality of a floater schedule after optimization, but also during optimization to determine what CSCs are problematic and should potentially be visited by a floater.

The second type of calculations is related to the floaters. The following inputs are used:

- List of CSCs to be visited by each floater (decision variable).
- Duration of stay (amount of work performed) by floaters at each CSC (decision variable).
- CSC characteristics (fixed parameters).

Using these inputs, the detailed floater schedules containing starting and finishing times at each CSC can be created. The feasibility of the schedules is also checked.

In Figure 21, an overview of the calculations is shown.

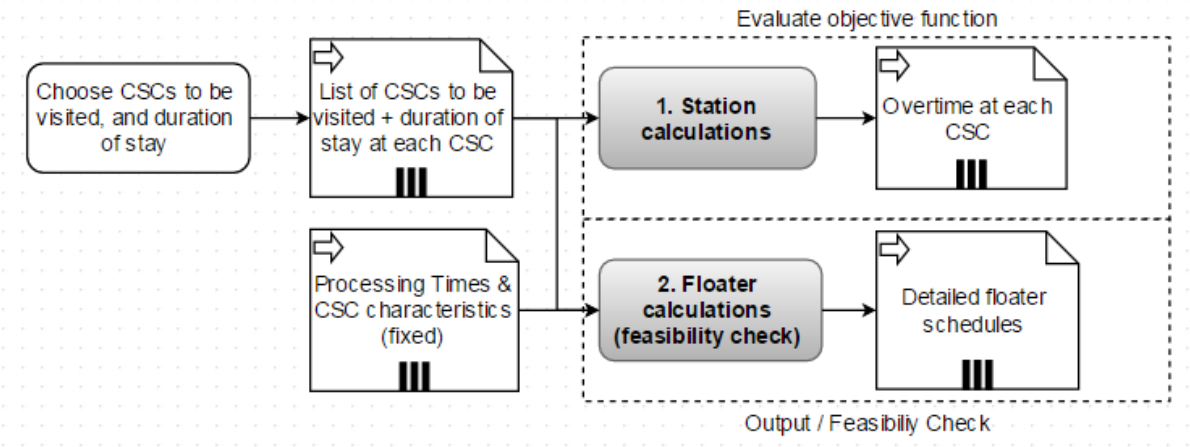


Figure 21. Calculation scheme

Station calculations

In order to accurately determine the overtime at a station on a specific car, the starting time at which this station can start its activities should be known. We assume that a station always starts its activities as early as possible. The assembly line under consideration uses non-conflicting overlaps and therefore the starting time is independent of the previous station. Thus, the only factor that influences the starting time of a station on a car is the finishing time of that station on the previous car.

In our calculations, we assume that a station starts its activities the moment the car enters the overlap area that precedes the station, the negative overlap. This is called the entry time $Entry_{ij}$ (see Section 4.2.7), which is known beforehand. The starting time of a station on a car is delayed by $Delay_{ij}$ if the station cannot start its activities at the start of the negative overlap.

In formula form, the starting time of activities on car i at station j is calculated as follows:

$$S_{ij} = Entry_{ij} + Delay_{ij}$$

To initiate the calculations, we note that a station can start processing its first car the moment it enters the negative overlap of the station, without delay:

$$Delay_{1j} = 0$$

The finishing time F_{ij} of car i on station j is calculated as:

$$F_{ij} = S_{ij} + \frac{p_{ij} - (R_{ij} * e)}{w_j}$$

Where p_{ij} denotes the processing time of car i on station j, and w_j the number of regular workers on that station (both fixed and known). R_{ij}denotes the duration of stay of a floater at this CSC (decision variable). The variable e is a number between 1 and 0 which indicates the floater efficiency. The product $(R_{ij} * e)$ indicates the amount of work that is done by the floater. In this thesis a value of 1 if assumed, meaning that a floater can perform the same amount of work as a regular worker in the same amount of time.

Workers move alongside the car while working on it, and move back towards the next car if the current car is finished or when they have reached the end of the positive overlap of the station (after which they can no longer work on the car). To clarify the calculation of the delay on a car, we use an example. In Figure 22 a station is shown, with entry and exit times.

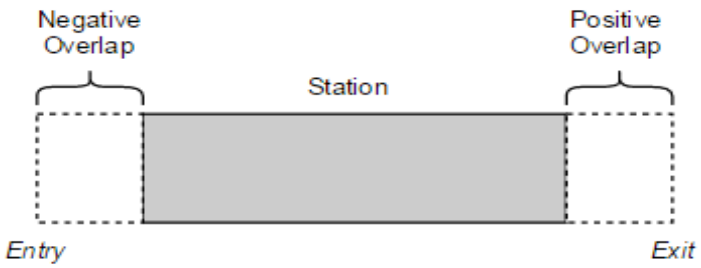


Figure 22. Station with entry and exit times

In Figure 23, the arrows indicate the starting time on car i+1 given the finishing time of the station on car i.

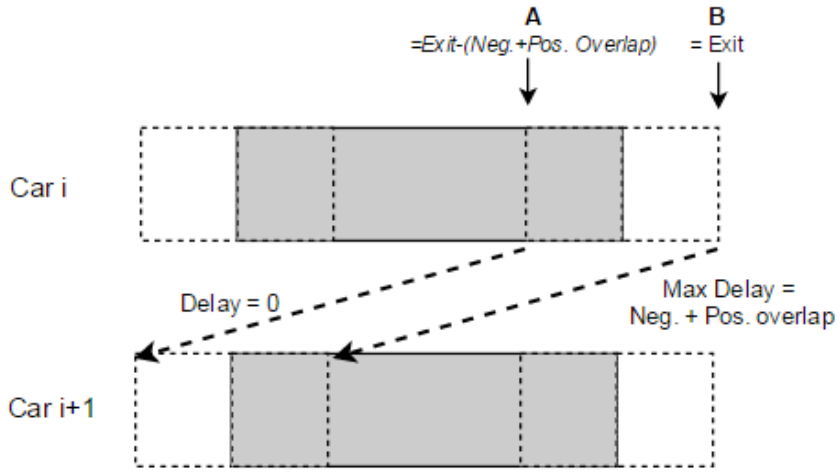


Figure 23. Starting time car i+1 given finishing time car i

To start as early as possible on car i+1, workers should finish their activities on car i at point A in Figure 23, which is equal to $Exit_{ij} - (Neg.Overlap_j + Pos.Overlap_j)$. The workers will move back to the end of the station either if they are finished working on the car, or when they have reached the end of the positive overlap, i.e., $Min(F_{ij}, Exit_{ij})$. The difference between these two times is the delay on the next car, which is either 0 or positive (a negative value would mean idle time). Thus, the delay on car i+1 at station j is given by:

$$Delay_{i+1,j} = Max\{0, Min(F_{ij}, Exit_{ij}) - (Exit_{ij} - (Neg.Overlap_j + Pos.Overlap_j))\}$$

Finally, the overtime OT_{ij} of car i at station j can be easily calculated from the finishing time F_{ij} as follows:

$$OT_{ij} = Max(F_{ij} - Exit_j, 0)$$

If a decision has been made on the amount of work R_{ij}performed by the floaters at each CSC, the formulas described above are used to calculate the starting/finishing times, delay, and overtime for each car.

Floater calculations

The formulas in the previous section 'Station calculations' are used to calculate the overtime on each CSC, given the amount of work that is performed by floaters. During their shift, floaters move through the assembly line to provide help at different CSCs. These CSCs form sequences which form the schedules together with the duration of stay at each CSC. In this section, we let SF_{kl} denote the starting time of floater k at the l^{th} CSC in his schedule.

A floater cannot start providing help at a CSC before the car has entered the station. Also, the floater must travel to the CSC and set up there, after finishing at the previous CSC at time $FF_{k,l-1}$. The time needed by a floater to 'move' from one CSC to the other is calculated as discussed in Section 4.2.8 and denoted by $t^{k}_{l-1,l}$ (here, l denotes the station associated with CSC l). The floater starting time SF_{kl} is given by:

$$SF_{kl} = Max\{\, Entry_{kl} \;,\; FF_{k,l-1} + t^{k}_{l-1,l} \,\}$$

The finishing time FF_{kl} for floater k at the ith CSC in his schedule is calculated as:

$$FF_{kl} = SF_{kl} + R_{kl}$$

Each floater can start his work at the first CSC he visits the moment the car enters the negative overlap at the station:

$$SF_{k1} = Entry_{k1}$$

This gives the starting point from which all other quantities can be calculated, using the two equations for SF_{kl} and FF_{kl}.

To determine if a schedule is feasible, we have to check that a floater can stay for the chosen duration of stay R_{kl} at the CSCs that he will visit. Travel time (discussed in Section 4.2.8) is the only restriction here. We have to establish that from the moment the floater is ready at a certain CSC, he has enough time to reach the second CSC and perform the chosen amount of work R_{kl} before the car leaves the (positive overlap) of the station. If a floater arrives 10 seconds before the car leaves the station, but has to resolve 20 seconds of overtime, the solution is infeasible. Floater k can visit the ith CSC in his schedule for the duration of R_{kl} if the following condition holds:

[Restriction 1]

$$SF_{kl} \leq Exit_{kl} - R_{kl}$$

This condition should hold for all floaters and CSCs visited, for the schedule to be feasible.

These formulas, together with those described in the previous section 'Station calculations', allow us to determine all variables necessary to perform optimization and determine detailed schedules.

To show the connection between the floater and station calculations, we give an example below (see Figure 24). The CSC list and the floater schedule are linked through the CSC numbers. The first CSC to be visited by the floater is CSC number 851. This number corresponds to the CSC with number 851 in the CSC list, where the characteristics of the CSC are stored (entry/exit time, processing time, etc.).

CSC List

CSC	Car	Station	EntryCar	ExitCar	StartCar	FinishCar	Proc	Delay	OT	Floater	R
850	5	49	628,60	691,40	628,60	649,38	20,78	0,00	0,00	0	0,00
851	6	49	681,40	744,20	681,40	738,91	65,51	0,00	0,00	1	8,00
852	7	49	734,20	797,00	738,91	800,82	61,91	4,71	3,82	0	0,00

Floater Schedule

CSC	851	3387	863	3393
EntryCar	681,40	1262,20	1315,00	1579,00
ExitCar	744,20	1377,80	1377,80	1694,60
R	8	14	15	34
StartFloat	681,40	1262,20	1350,61	1579,00
FinishFloat	689,40	1276,20	1365,61	1613,00
Station	49	51	49	51

Figure 24. Example showing the connection between floater schedules and CSCs

4.4 Solution Approach

In this section, we propose our solution method. First we discuss the choice to divide the total assembly line problem into sub problems that each deal with a part of the assembly line. We remove the duration of stay as a decision variable by introducing 'Duration of Stay' and 'Visit' rules. We then discuss why an approximate method is needed instead of an exact approach. Finally, we improve the starting solutions using heuristics, namely Simulated Annealing and a Genetic Algorithm.

4.4.1 Problem subdivision into Working Areas

In Section 2.1 we mentioned the subdivision of the total assembly line into 'lines'. We now discuss some properties of these lines and the consequences for our solution method.

- To work at a station, a floater has to possess certain competences which can differ between each station. These competences have to be trained, and the workers and floaters at Company A are trained to have all the competences needed for a line, for example the chassis line.
- The role of floaters is often filled by team leaders. These team leaders are assigned to a line for which they are responsible.

Schedules in which a single floater provides help at different lines are undesirable for Company A. For our solution method to match the real life situation, it should follow the current division of the assembly line. Besides making the produced schedules applicable in practice, there are some other benefits to this approach:

- By defining working areas for floaters to be the same as the lines, competences can be managed. If it turns out that an extra floater for an area is desired, it is clear what competences must be trained to achieve this. Within an area, competence restrictions are removed from the model which limits the complexity of the problem.
- Choosing floater working areas identical to the production lines also assure that these competences are related, as all operations performed on a single line require a similar skill set. This might increase the performance of floaters, as they will gain experience in performing a certain type of operation. One could also argue that job satisfaction will increase among employees when they perform operations that are related to one another, which was also found by Agrawal (1985). Job satisfaction may be further enhanced by the fact that floaters will interact with the same regular workers more often, allowing for social bonding.

For these reasons the decision has been made to divide the total final assembly line into smaller areas based on the current division of the assembly line into 'Lines' (Section 2.1), such as the Chassis line and the Door line.

Each line is a *working area* for a (group of) floater(s). The number of available floaters will be defined as a parameter for the model. Schedules will be generated for all floaters for each area consecutively. Table 6 and Figure 25 show the division of the final assembly into the working areas.

Line	Working Area
Body line	1
Skillet line 1	2
Skillet line 2	3
Skillet line 3	4
Skillet line 4	5
Window line	6
Chassis line	7
Engine line	8
Double door line	9
Door line	10
Single Pallet Line	11

Table 6. Numbering of working areas

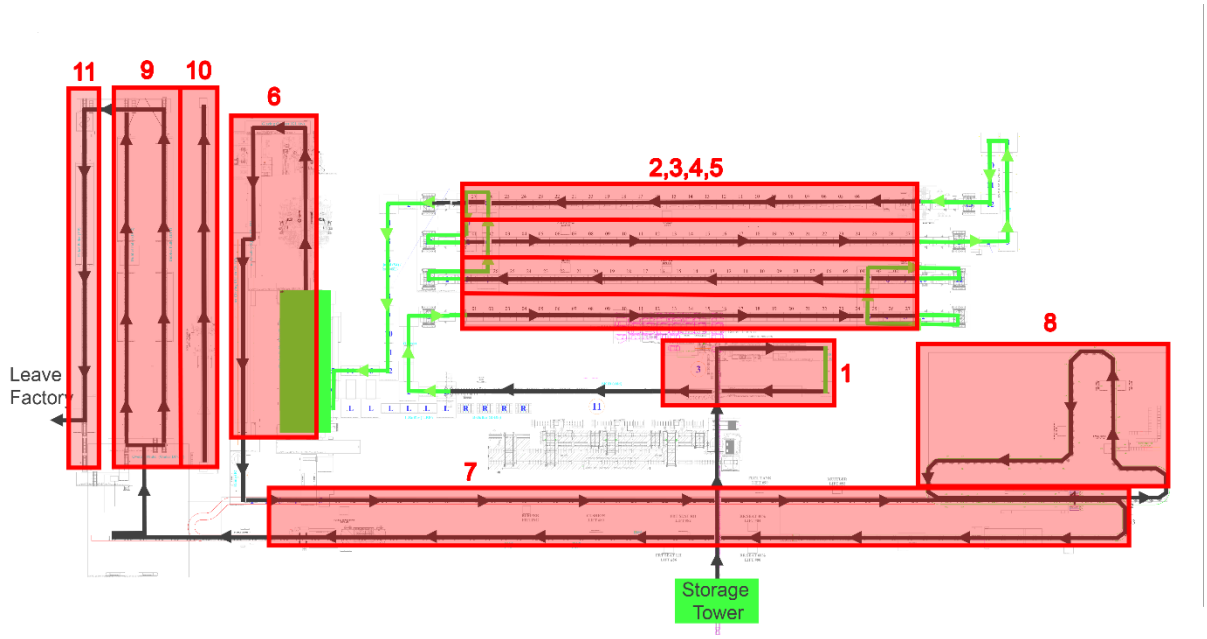


Figure 25. Division of assembly area into working areas

4.4.2 Duration-of-Stay and Visit Rules

As discussed earlier, there are two questions that need to be answered to provide a solution to the problem (schedule). These questions are:

- Visits: At which CSCs should each floater provide help (considering travel time restrictions)?
- Duration of stay: How long should a floater stay at each CSC?

If we arbitrarily choose the duration of stay at a CSC, we are left with a large amount of options to choose from. Say we want a CSC to be visited by a floater. Keeping the duration of stay an integer number of seconds, we can still choose to visit the CSC anywhere between 1 second and the total time the car is at the station. Every different option has a different finishing time of the floater at the CSC, and the resolved overtime also changes.

There are some practical/logical limitations to the duration of stay. A floater can only stay at a CSC from the time he arrives until the car leaves the station. If the floater arrives at an assigned CSC before the car enters the station, he has to wait for the car to arrive (which is possible). To further limit the search space, we have constructed three rules that will determine the duration of stay. By selecting a Duration of Stay rule (DOS-rule), **we limit the set of decision variables to a list of CSCs to be visited.** We propose the following **DOS-rules**:

1. Stay until all overtime is resolved (Area 1 in Figure 26).
2. Stay until the station can start its activities on the next car at the regular station bound (Area 1 + 2 in Figure 26).
3. Stay until the station can start its activities on the next car at the earliest starting time (at the start of the negative overlap). This amount of time is indicated by the sum of area 1, 2, and 3 in Figure 26.

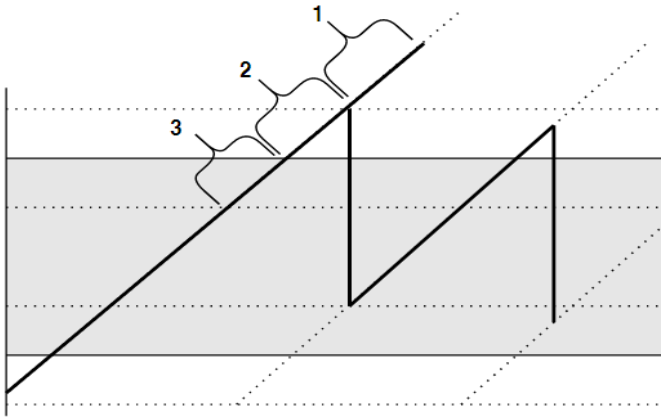


Figure 26. Duration of Stay (DOS) rules

When considering adding a CSC to the schedule of a floater, we calculate the duration of stay according to the rules listed above. This is a strive value, it is possible that the calculated duration of stay is shorter than the actual available time for the floater at that CSC. For example if the floater arrives 20 seconds before the car leaves the station, but has to perform 30 seconds of work according to the selected DOS-rule. We propose two **Visit-rules** to determine the final duration of stay:

1. The floater stays for as long as possible (according to Restriction 1 in Section 4.3.3) until the prescribed duration of stay is done or he runs out of time.
2. The floater only visits the CSC if the prescribed duration of stay can be fully met.

Even though the DOS and Visit rules do not guarantee an optimal duration of stay, they enable us to choose the CSCs to be visited. In Chapter 5 we investigate the influence of the DOS and Visit-rules on the obtained solution quality.

4.4.3 Solution Space

By defining the Duration-of-Stay and Visit rules in Section 4.4.2, we have limited the set of decision variables to a list of CSCs to be visited by each floater plus a selection of DOS and Visit rules. For every CSC a decision has to be made whether or not a floater will be assigned to help. Making this decision results in a schedule, which consists of a list of CSCs to be visited by each floater together with a duration of stay at each CSC.

The number of possible choices is finite: there is a finite number of CSCs to be chosen to be visited by each floater. Therefore, selecting CSCs to be visited by each floater in a way that minimizes overtime and defects is a combinatorial optimization problem. To find out if enumerating all possible solutions is a viable approach, we consider the solution space size.

Consider a case withf 10 cars and 3 stations, which gives 30 CSCs. If we have 2 floaters available, we decide for each CSCs if it will be visited by one of the floaters (decision 1 or 2) or by none of the floaters (decision 0). Thus, we have 3 possible choices for each CSC, resulting in 3^30 = 3.486.784.401 possible combinations. This is an upper limit to the number of possible choices.

Realistically, a floater can only be at one station at a time. This example problem has 10 cars, so it takes roughly 10 takts to process them all. For each takt a decision has to be made at which station a floater will provide help. For 3 stations, the number of choices is 4 (Visit station 1, 2, 3 or none) per takt (for one floater). So for each floater we have a number of possible solutions of $(Stations+1)^{\#Takts}$. For two floaters, this gives us a total of $\#floaters*(Stations+1)^{\#Takts}$ = 2*(3+1)^10 = 2.097.152 possible solutions. This number grows so quickly, that for a problem with 2 floaters, 5 stations and 150 cars, the number of possible combinations is larger than there are atoms in the observable universe (generally approximated to be around 10^80 atoms). Feasibility constraints will reduce the number of possible solutions, but it still grows exponentially with the number of cars that we consider. Therefore we conclude that enumeration is not possible for the problem of selecting CSCs to be visited by each floater.

4.4.4 Candidate CSCs

When considering what CSCs to visit, it is possible to limit the search space by only considering those that have or cause overtime in the situation without floaters. The effect of overtime of a car on future cars is limited because of station limits. The maximum delay on the next car is equal to the overlap, and the maximum 'lead' a station can have on the regular takt is also equal to the overlap.

We therefore propose to reduce the total list of CSCs, to a list of *candidate CSCs* that consists of the CSCs that have overtime and a fixed number of CSCs that precede them. This number should be chosen equal to the maximum number of CSCs that have a delayed start before overtime occurs. We refer to this number as '*candidate string length*'.

For the problem at Company A discussed in this thesis, a dataset is available for 10 workdays consisting of two 8 hour shifts. This dataset contains all cars that were produced, their configuration, and processing times. In our analysis of this dataset it became clear that when overtime occurred, a maximum of two preceding cars started late (e.g., with a 'delay' as specified in Section 4.3.4). This means that overtime was never caused by a car that passed the station more than 2 takts ago. We therefore set the length of the candidate string at 2 for the experiments carried out in this thesis. For other applications, this number should be adjusted accordingly. Setting the string length too short might prevent some good solutions from being found. Setting the string length too long will unnecessarily increase the solution space size, which causes CPU time to be wasted.

4.4.5 Proposed Solution Method

We now propose our solution method to create and optimize floater schedules (see Figure 27). This process is repeated for each working area.

1. First we create an initial solution using a constructive heuristic.

We do this using the procedure described in Section 4.5. If any overtime remains in the area, we go the next step.

2. We select a CSC with overtime that is not visited by a floater, from the list of candidate CSCs.

This selection is guided by a set of rules, and focuses on the CSC with the most overtime. These rules are discussed in more detail in Section 4.6.1. After a CSC has been selected, we perform the following step:

3. Optimize the floater schedules in a time window (optimization window) around this CSC, using metaheuristics.

This means that we iteratively optimize small parts of the total floater schedules (partial schedules). The main idea behind this, is that overtime at a CSC can only be resolved by changing the floater schedules around the moment of occurrence of that CSC. For example, the way in which floaters are scheduled at the beginning of the shift has no influence on CSCs that occur towards the end of the shift. As discussed in Section 4.4.4, overtime at a station is never caused by a car passed the station more than 2 takts ago. The metaheuristics in step 3 are designed to change the floater schedules in a way that minimizes overtime and defects. When the optimization process at step 3 has finished, we move on to step 4.

4. The partial schedules resulting from step 3 are inserted back into the main floater schedules. The overtime of all CSCs is recalculated.

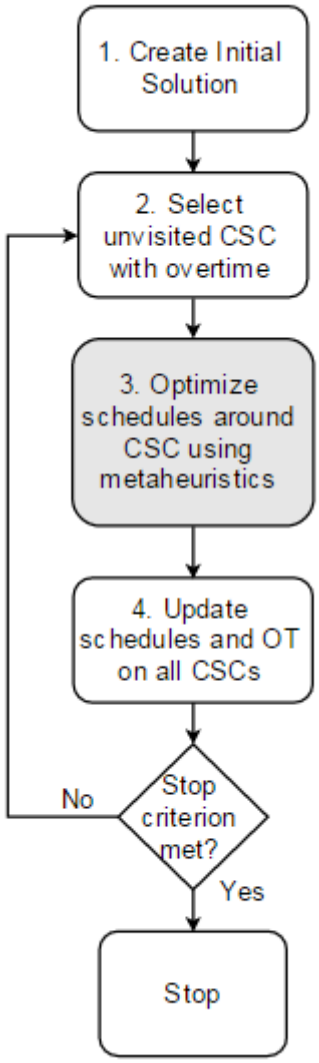


Figure 27. Solution Method

After this step, we evaluate the stopping criteria. If all overtime has been resolved, or the assigned running time has expired, we stop. Additional stopping criteria are discussed in Section 4.6.1. If no stopping criterion is met, we move back to step 2.

The main idea is that we iteratively take out a part of the floater schedules, optimize this partial schedule and then insert it back into the general schedule. We basically perform a local search process at step 2 (selecting a CSC), that iteratively calls another local search process at step 3 (optimizing the partial schedule around that CSC). This process is illustrated in Figure 28, which uses the same format as Figure 20 in Section 4.3.1.

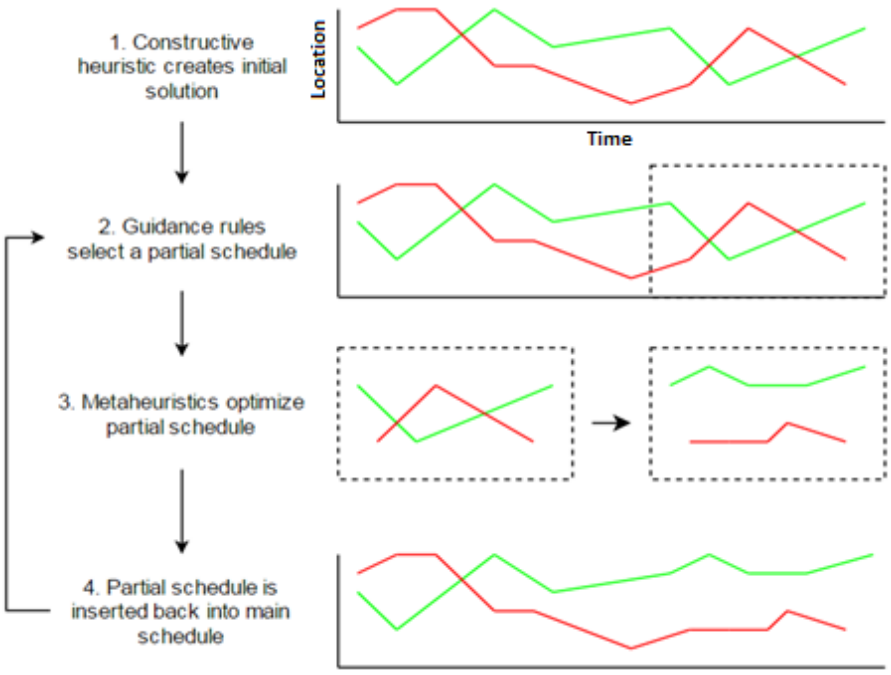


Figure 28. Solution method process

Motivation for selected metaheuristics

The problem of optimizing (partial) schedules (step 3 in Figure 27 and 28) is a combinatorial optimization problem, in which we search for an optimal solution (list of CSCs to be visited by each floater) in a finite solution space. The size of the solution space as discussed in Section 4.4.3, makes complete enumeration of our problem impossible. The search space at step 3 in our solution method is smaller, but even for a relatively small time window we expect that the number of candidate CSCs will quickly become too large to fully enumerate the search space (for our problem, a time window of 2 takts already contains around 20 CSCs on average). Therefore an approximate method is needed. Meta-heuristics (discussed in Section 3.6) are used to find sufficiently good solutions to combinatorial optimization problems. Meta heuristics come in different shapes and sizes. Because our problem does not classify as a standard problem known in the literature (such as the Vehicle Routing Problem), it is difficult to tell beforehand which meta-heuristic is best suited for solving our problem.

Therefore we implement two metaheuristics, namely Simulated Annealing (SA) and a Genetic Algorithm (GA) (first discussed in Section 3.6), which are widely used for solving various problems. We have chosen these algorithms because of their different approach to the optimization process. SA is a single solution based metaheuristic, which stores and alters a single solution during the optimization process. In contrast, GA uses a population of solutions in its optimization process. The two heuristics also differ in the way new solutions are created: SA mutates a (small part) of the solution, whereas GA recombines (good) solutions to create new solutions. GA has the additional benefit that the crossover operator allows to combine the first part of a schedule with the second part of another schedule (see Section 5.3.2). This might turn out to be beneficial as the problem in this thesis allows for ‘good’ parts of schedules to be combined without these parts affecting each other (to a large extent).

We deemed Tabu Search (TS) less suitable for the optimization of partial schedules. TS selects the best neighbor solution, unless this is forbidden by the Tabu list. TS prevents cyclic behavior, which we do not expect to occur to a large extent in our problem. If we move to a new solution, for example by switching a CSC from floater 1 to floater 2, we do not expect the same reverse move to be the best option in the next step. Also, the number of moves available at each point in the optimization process will be so large that it will become difficult to guide the search process towards good areas in the solution space in a consistent way.

We optimize the parameter settings of both the SA and GA metaheuristics on our problem (see Section 6.3), and then compare which one works best. Additionally, we implement a simple Hill Climbing (HC) algorithm, which iteratively changes a solution and accepts it if it improves the objective value. We compare the results of SA and GA to those acquired by HC, to gain insight in the benefit of a 'smart' optimization process over a 'regular' hill climbing algorithm.

4.5 Initial Solution

We now propose constructive heuristics that create initial floater schedules by selecting CSCs one by one and adding them to the schedules.

4.5.1 Process

The mechanics of overtime and floaters have been discussed and defined in Sections 4.2 and 4.3. Based on this, we now focus on the higher level decisions on what CSCs to visit and for how long. The detailed schedules can be derived from these decisions.

So far, we have constructed a model in Section 4.3 in which we formulated restrictions and calculations for our problem. We have subdivided the problem into areas in Section 4.4.1. Now we propose constructive heuristics to provide starting solutions. The complete process of constructing an initial solution is shown in Figure 29. This process corresponds with step 2 of the total solution method in Figure 27.

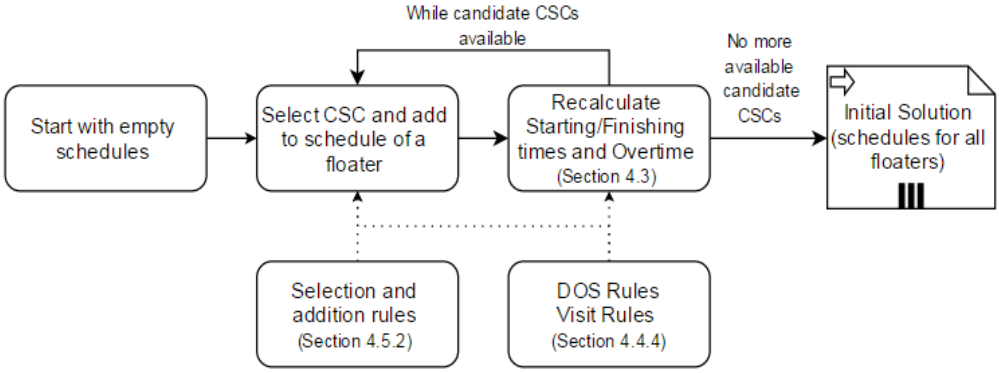


Figure 29. Constructive heuristic

Selection and addition rules prescribe which CSC will be added to the floaters' schedule next. The DOS and Visit rules then determine for how long the floater will stay at the CSCs. The DOS and Visit rules are also used during the selection of CSCs, as the amount of overtime that can be resolved at a CSC depends on the duration of stay. After adding a CSC to the schedule of a floater, all quantities are recalculated using the formulas from Section 4.3.3. The overtime on each CSC is updated, so the constructive heuristic has the most accurate information available when making a decision on which CSC to add next. This process repeats until no more candidates are available. This occurs when the floaters cannot reach another candidate CSC within the planning horizon (shift) after they finish their activities at the last CSC in their schedule, or when there are none left.

We implement the initial solution process both sequentially (construct routes for the floaters one by one) and in parallel (construct schedules for all floaters simultaneously) where possible. When considering the next CSC to be added to one of the schedules in the parallel process, it can then be assigned to the floater that is in the best position (time and location wise) to provide help.

4.5.2 Selection and addition rules

We now discuss selection and addition rules that are used to decide which CSC is added to the floater schedules next. The first two rules, First Come First Serve (FCFS) and Time Oriented Nearest Neighbor (TO-NN), stem from literature and are based on the time element of the problem. Both will be implemented sequentially and in parallel. Next to FCFS and TO-NN, we propose a generic 'Biggest Problem First' algorithm.

For the first two (time based) heuristics (FCFS and TO-NN), we start at the earliest moment in time, which is the time the first car enters the first station of the area. We then select a CSC to be visited next, based on some criteria. After selecting a CSC, overtime and floater schedules are recalculated. By adding a CSC to the schedule of a floater, this floater moves forward in time.

FCFS

A simple time-based selection rule for 'jobs' commonly found in scheduling literature is the First Come First Serve principle. Applied to the problem in this thesis, the FCFS rule selects the earliest CSC with overtime in every iteration and tries to add it to the schedule of a floater, until no more CSCs are available. The sequential implementation of FCFS performs this process for one floater at a time. The parallel version will consider all floaters when trying to add a CSC to the floater schedules.

Time Oriented Nearest Neighbor (TO-NN)

The next heuristic we propose is based on the Nearest Neighbor heuristic commonly found in Traveling Salesman Problem literature. The NN heuristic starts at a selected city, and then travels to the city that is closest to the current city. 'Close' originally meant the smallest distance, but researchers have added other criteria to determine proximity. For example, Solomon (1987) added temporal proximity as a criterion for a TSP problem with time windows. If we are currently at city i, and consider visiting city j, Solomon (1987) used the following criteria: direct distance from i to j, the time difference between completion of service at i and beginning of service at j, and the urgency of delivery to customer j (time remaining until latest possible service start at t).

We propose a modified version of the Time Oriented Nearest Neighbor approach of Solomon, as time is the main limitation in our problem and distance can be directly translated into time. If we want to optimally resolve overtime, we should make decisions at every stage that will resolve as much overtime as possible while investing as little time as possible. We therefore score the CSCs for overtime solved per second invested. In formula form:

$$Score = \frac{a1 * \ Overtime\ resolved}{a2 * (Travel\ time + Idle\ time + Duration\ of\ stay)}$$

The weights a1 and a2 can be adjusted to rescale the quantities. An analysis of the effect of these weights on the solution quality can be found in Chapter 5. In each step, the CSC with the highest score is selected. The TO-NN heuristic is implemented both sequentially and in parallel.

Biggest Problem First

This constructive heuristic will sort the list of CSCs with the most problematic on the top, with overtime as criterion. It will then try to insert each CSC to a floater schedule, starting from the top. If a CSC cannot be inserted to one of the schedules, the heuristic will move on to the next CSC. The Biggest First method creates schedules for all floater simultaneously (parallel approach).

In total 3 heuristics (plus variants) for obtaining an initial solution have been proposed, which are shown in Table 7.

Heuristic	Variants	Selection of CSCs based on:
First Come First Serve (FCFS)	Sequential Parallel	Time of occurrence of CSC
Time Oriented Nearest Neighbor (TO-NN)	Sequential Parallel	Overtime resolved per second invested
Biggest Problem First (BPF)	Parallel	Overtime at CSC

Table 7. Overview of constructive heuristics

4.6 Optimization

After we have obtained a starting solution using the constructive method discussed in Section 4.5.1, we further optimize the schedules with the goal of minimizing the objective function, i.e., resolving as much overtime and defects as possible (see Section 4.3.2 for the objective function).

4.6.1 Approach and Guidance Rules

After constructing a starting solution, we are left with problematic CSCs which have overtime and are not visited by a floater. Improving the objective function is only possible by (partially) resolving the overtime on these CSCs. To do so, we need to change the parts of the schedules in which these problematic CSCs occur, which we call optimization windows, as shown in Figure 30.

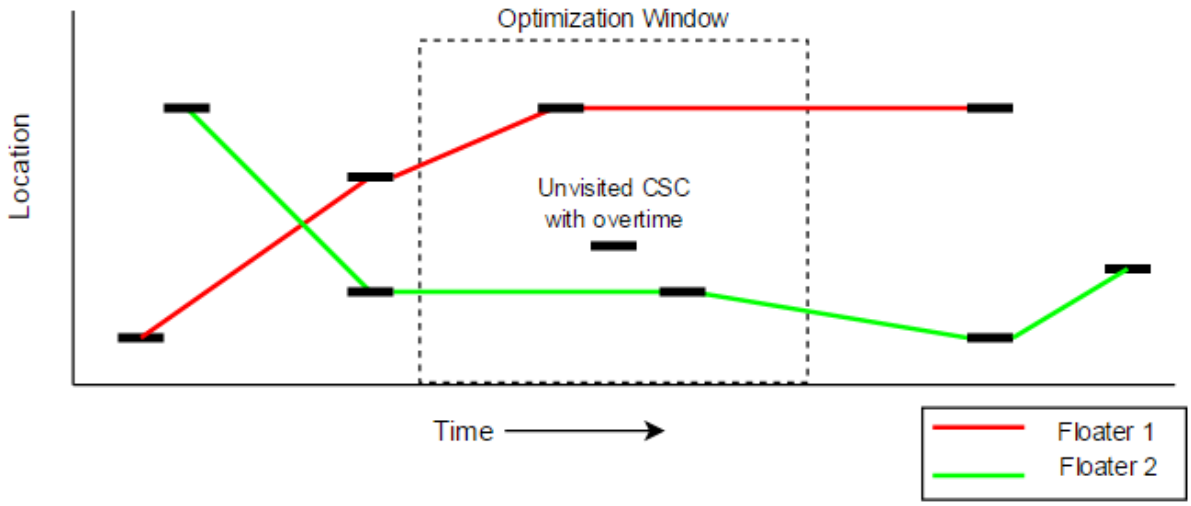


Figure 30. Problematic CSC and Optimization Window

Guidance rules for the search process

Our solution method proposed in Section 4.4.5, iteratively selects a problematic CSCs around which the schedule will be optimized.

The first method we propose to do this, is to sort the candidate CSCs once on overtime (in decreasing order) at the start. The CSCs are then selected starting with the top CSC, and moving down the list. The CSCs are not sorted on overtime during this process. This is our first guidance rule.

The second guidance rule we propose sorts the candidate CSCs every time a partial schedule has been optimized, and selects the CSC with the most overtime. This is done because the overtime on the CSCs changes as the partial schedules are optimized. To prevent the search process getting stuck by selecting the same CSCs over and over again, we add a Tabu list that stores which CSCs have been selected recently. Once a CSC has been selected, it is not allowed to be selected again for a predetermined number of iterations. This number should be large enough to prevent cycling, but small enough to guide the search process to promising CSCs.

This leaves us with two **guidance rules**:

1. Sort the candidate CSCs on overtime once, and select CSCs from top to bottom
2. Sort the candidate CSCs on overtime, and select the top CSC to optimize the partial schedule around it. After each partial schedule has been optimized, update the overtime on all candidate CSCs, sort the list, and select the top-most CSC that is not on the Tabu list.

An analysis of the performance of the guidance rules can be found in Chapter 5.

Process

We now propose an optimization plan for the partial schedules (schedules within the optimization window in Figure 30). After a CSC has been selected according to the higher-level guidance rules, we perform the following steps:

1. Copy the floater schedules in a time window around this CSC (the width of this window is a parameter, for example from 2 minutes before until 2 minutes after the CSC occur).
2. Use an optimization heuristic to try to find a better partial schedule for this time window.
3. Insert the partial schedule back into the main schedule.

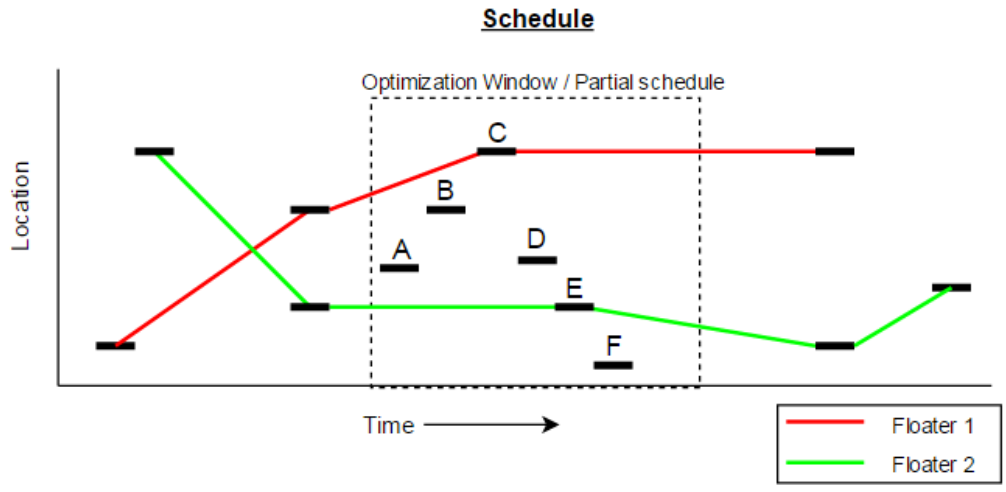
In Section 4.6.2 and Section 4.6.3 we propose metaheuristics to perform step 2 of this process.

Optimization window

It is important to note that in the optimization window in Figure 30, only the CSCs that are visited by floaters and CSCs with overtime are shown. Inside the optimization window, there are other CSCs without overtime that can be visited, which might help to resolve the overtime. These CSCs, together with the CSCs that have overtime, are the 'candidate' CSCs, as discussed in Section 4.4.5. The width of the optimization window is a parameter. Choosing it too narrow could result in missed opportunities (smarter routes could be formed), but too wide could result in a search space that is so large that it becomes hard to find good solutions. An analysis of the optimization window width can be found in Chapter 5.

Problem notation

The optimization process of a partial schedule starts by creating the notation string that will be used by the metaheuristics. This is done as follows. A list is gathered of all candidate CSCs in the time window (all CSCs which cause or have overtime). These CSCs are sorted in time, resulting in the notation string as shown in Figure 31. For each CSC in the notation string, the number indicates by which floater it is visited.



Resulting notation string

CSC	A	B	C	D	E	F
Floater	0	0	1	0	2	0

Figure 31. Translation of partial schedule into problem notation

Translation back to schedules

For each candidate CSC in the notation string, the optimization heuristic will choose whether it is visited by a floater, and if so, by which one. To evaluate the objective value of a solution (and to insert the partial schedule back into the mains schedule), the notation string used by the algorithms has to be translated back into a schedule. Starting from the left of the string, routes are constructed by adding each CSC to a partial floater schedule, if a floater has been selected for that CSC. The duration of stay at each CSC is again determined by the DOS and Visit rules (see Section 4.4.2).

Limitations of the used problem notation

The problem notation introduced in the two previous paragraphs, does not capture all information embedded in a schedule. In Figure 31, none of the candidate CSCs occurred at the same moment. Now consider the example in Figure 32.

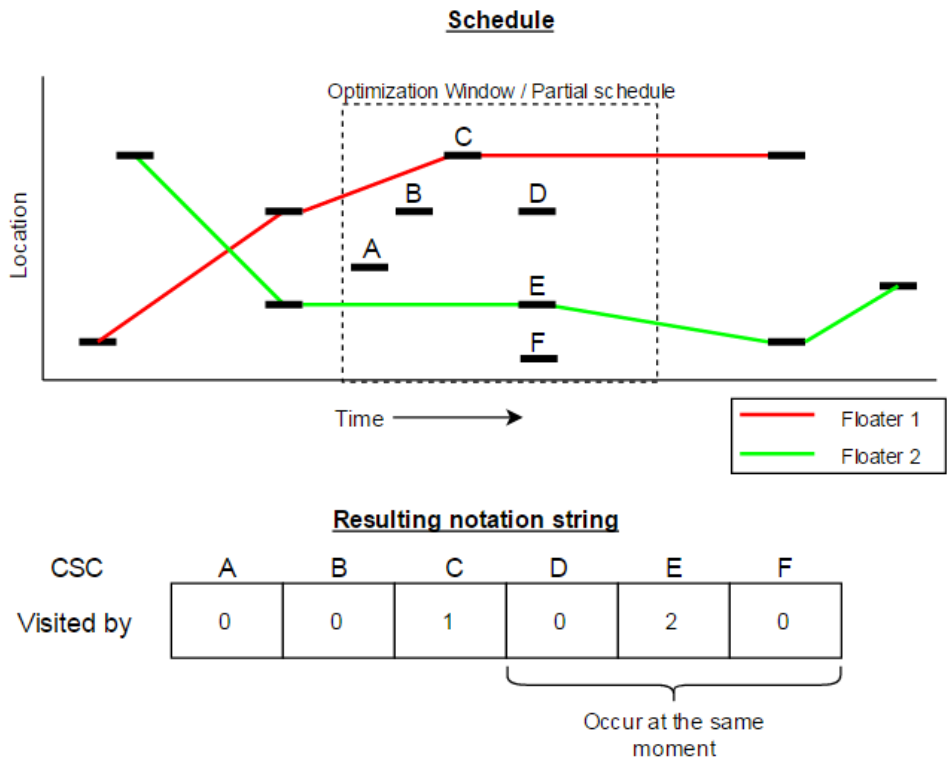


Figure 32. Problem notation for simultaneously occurring CSCs

The CSCs indicated by D, E and F occur at the same moment. In our method, these CSCs are put in a sequence based on their station number when the notation string is created. We realize that this excludes some solutions to our problem, as the sequence in which they will be visited will always be the same. In the example above, if the same floater is assigned to CSC D and E, he will always visit CSC D first. Due to time restrictions on the execution of this thesis, we have left the sequencing of these CSCs that occur at the same moment out of consideration. In Section 6.1.1 we review to what degree this occurs; the resulting effect on the performance of our method; and the implications for application of our method in assembly lines with a different configuration.

4.6.2 Simulated Annealing

The first proposed heuristic for optimizing the partial schedules is Simulated Annealing (SA). In Section 3.6.1 we discussed the fundamentals of SA. In our application, a 'solution' is a partial schedule.

Neighbor structure / Initial number of changes

We use the following neighbor structure to create each neighbor. For a set *number of changes*, a random candidate CSC in the notation string is selected. This CSC is then either swapped between floaters, or moved in or out of the schedule depending on whether or not it is visited (if the CSC is already visited, it will be taken out of the schedules). An example of these changes is given in Figure 33, for a partial schedule for two floaters. A zero denotes that the CSC is not visited by a floater, and a 1 or 2 indicates that the CSC is visited by floater 1 or 2 respectively.

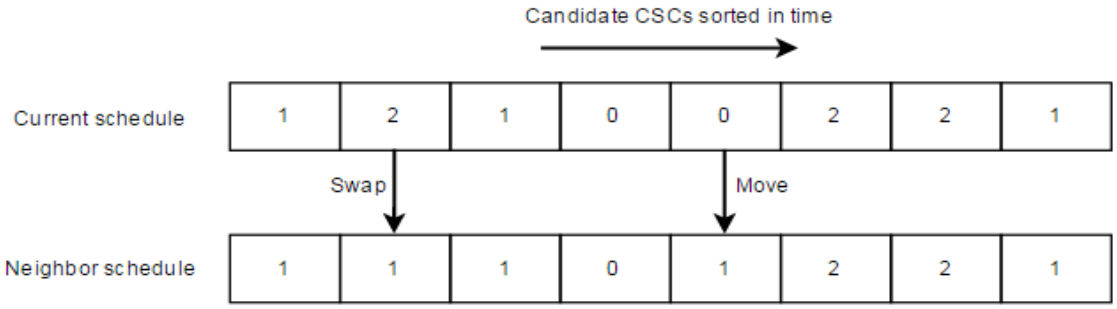


Figure 33. Neighbor creation in Simulated Annealing

The number of changes is a parameter. We set the *initial number of changes* as a parameter, and let it decrease to 1 in a linear fashion over the total number of iterations.

Cooling schedule

At each iteration i (at each temperature), a number of neighbor solutions is created. For each neighbor (partial schedule) that is created, the objective value is calculated. If the objective value is at least as good as the current solution it is accepted. If it is worse (i.e., higher value) it is accepted with a probability of $e^{-(neighbor\ obj.\ value - current\ obj.\ value)/Temperature}$. This chance approaches 0 as the temperature moves towards 0, and small increases in the objective function are accepted more often than large increases. The temperature is determined by the cooling schedule. Initially, schedules that deteriorate the objective function have a high probability of being accepted. As SA progresses and the temperature decreases, worse solutions are accepted less often.

A cooling schedule consists of the following elements: a starting temperature (T_{start}), a number of neighbors (N) to be created at each temperature, a temperature function $T(i)$ and a stopping criterion (Eglese, 1990). We choose a starting temperature T_{start}, such that initially all neighbor moves will be accepted. We choose to run the SA algorithm for a set number of iterations i_{max}. This allows us to adjust the amount of CPU time that is assigned to optimizing each partial schedule. A longer run time will produce better partial schedules, but limits the number of partial schedules that can be optimized in a fixed amount of time. A detailed analysis of this tradeoff can be found in Chapter 5. Because we limit the SA run time by i_{max}., we do not employ other stopping criteria. When the SA search process ends, the best solution (partial schedule) is returned and inserted back into the main floater schedules.

The temperature function determines the temperature at any given moment in the SA process. Eglese (1990) discussed the different kind of temperature functions implemented by researchers. They distinguish between functions with and without feedback. Those without feedback calculate the temperature based on the number

of iterations i or running time and the starting temperature. Functions with feedback take information gathered during the optimization process into account for determining the temperature. Eglese (1990) state that functions with feedback had long running times in practice, or relatively poor solutions when the running time is limited. Therefore we implement two temperature functions without feedback, as shown in Figure 34.

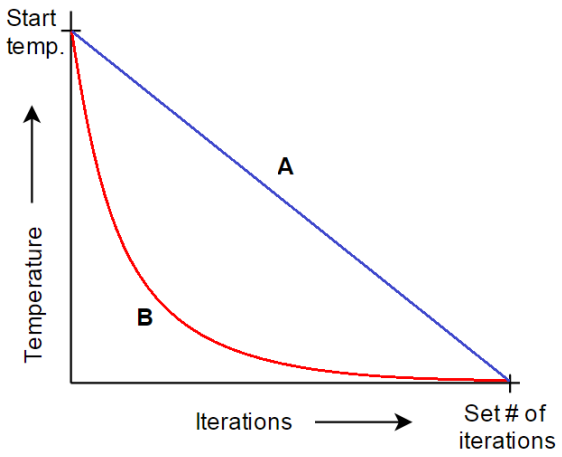


Figure 34. Cooling schedules for Simulated Annealing

For temperature function A (see Figure 34) the temperature decreases in a linear fashion, becoming nearly 0 at the set number of iterations, giving a temperature function of:

$$T(i) = T_{start} - \left(\frac{T_{start}}{i_{max}}\right) * i + 0.1 \qquad \text{[Temperature function A]}$$

The second temperature function (shown by line 'B' in Figure 34) is the exponential temperature function originally proposed by Kirkpatrick et. Al (1983). This temperature function is given by:

$$T(i) = T_{start} * \theta^{i} \qquad \text{[Temperature function B]}$$

In this function, the cooling rate θ determines the pace of the cooling, which should be chosen between 0 and 1, but is typically close to 1 (Nourani and Andresen, 1998). We determine the value of this parameter by first setting a fixed number of iterations and a starting temperature, and then choosing a value of θ that gives the desired temperature curve (see Chapter 5). Temperature function B results in a temperature (together with the chance of worse solutions being accepted) closer to 0 in an earlier stage of the optimization process, allowing for a longer stage of 'solidification' at the end of the process. Both the linear and exponential temperature functions have been widely used since they were introduced (Nourani and Andresen, 1998).

Parameters

Concluding, the following parameters can be changed for the SA algorithm:

- Number of iterations *i* (number of temperature changes)
- Number of neighbors *N* created at each iteration.
- Starting temperature (T_{start}).
- Temperature function (option A or B in Figure 34).
- Cooling rate θ (for cooling schedule B)
- Initial number of changes (the initial number of swaps/moves per neighbor).

4.6.3 Genetic Algorithm

The second heuristic we propose is known as a Genetic Algorithm. We discussed the fundamentals of GAs in Section 3.5.6, and will now discuss how we apply a GA to the problem in this thesis. GAs work on a set of solutions called a population. In our application, each individual is a solution/partial schedule. Each CSC in the notation string corresponds with the term 'gene' in GAs. New solutions are created using crossovers and mutations (events that occur in genetics) while searching for a better solution.

Overall process

The GA process of generating new populations is shown in Figure 35. From a population, some individuals are selected to produce offspring based on their fitness value. These parents are placed in an intermediate population. Next, crossovers are applied to these parents to produce offspring. The next generation is finalized by applying mutations. We chose to implement both 'classical' generational replacement, and an elitist approach as discussed in Section 3.6.3. In the classical approach, the newly created offspring replaces the previous population entirely. In the elitist approach, both parents and offspring are combined into a single population, and the top half is selected as the new population (see Figure 35).

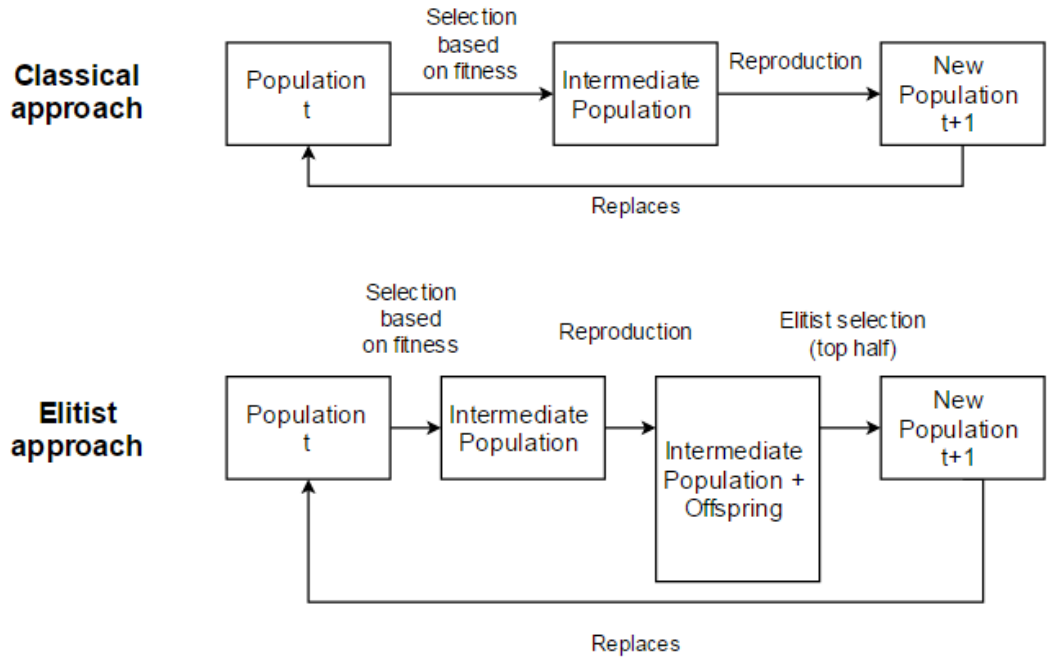


Figure 35. Classical and Elitist Genetic Algorithm approach

This process then repeats until a set number of generations has been completed: in each generation parents are selected based on fitness value, and a new population is generated.

Initial population

Applied to the problem in this thesis, the Genetic Algorithm will start with the current partial schedule as one member of the initial population. The initial population is further filled with randomly modified versions of this partial schedules, ranging from slightly (1 CSC/gene mutated) to more heavily modified versions (100% of genes mutated).

Reproduction

From this population pairs of solutions are selected as parents to produce offspring, based on their fitness value. With a predefined chance $P_{crossover}$, these parents produce offspring solutions by randomly selecting a crossover point and performing a crossover operation, which results in two 'children' (see Figure 36). $P_{crossover}$ is

a parameter in our model, in the literature its value is kept between 0.6 and 1. If no crossover is performed, two duplicates of the parents are put in the offspring population.

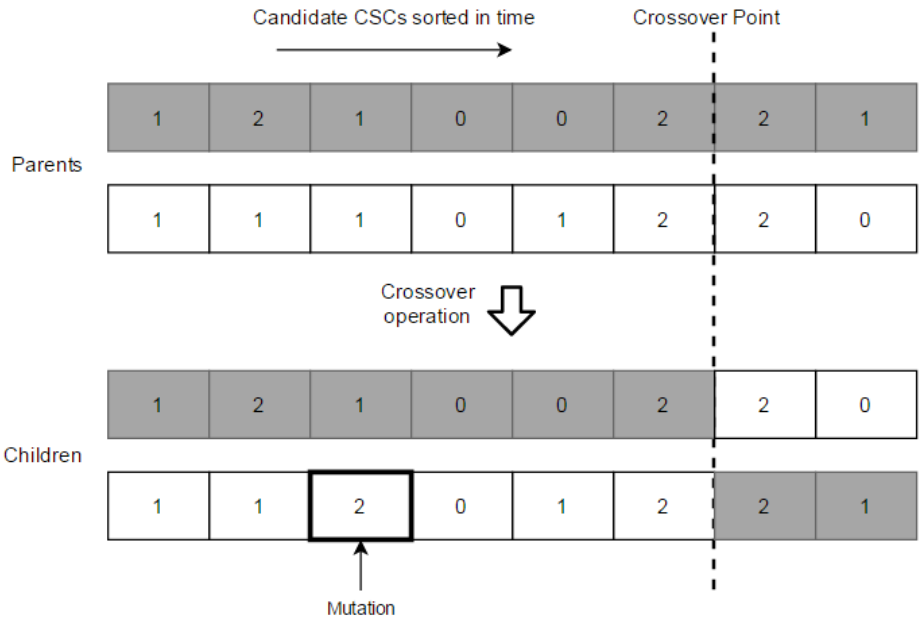


Figure 36. Offspring creation in Genetic Algorithm

This process is repeated multiple times, until the desired offspring population size has been reached. To complete the reproductive process, a mutation operator is used to ensure diversification in the search process. Each gene of every individual has a chance $P_{mutation}$ to be mutated. This is done by randomly changing the value of the gene (respective block in the notation string) (see Figure 36). We set the value for $P_{mutation}$ equal to 1/l, where l is the notation string length, as suggested by Bäck (1993).

Fitness function

For each member of the newly generated offspring population, the fitness value is calculated, which is equal to the objective function. We use the objective function discussed in Section 4.3.2 as fitness function. Based on the fitness value, members of the population will be selected to be put in the next intermediate generation (see Figure 35). Members of the population with a better fitness function have a higher change of being selected, and each member can be selected more than once. The intermediate generation then produces offspring as discussed in the 'reproduction' section earlier.

Selection of parents

Selection methods calculate a probability p_i for each individual in the population to be selected as parent. Based on p_i, stochastic selection (sampling) is performed which can be done in multiple ways. We use the ´stochastic universal sampling´ procedure, which is theoretically perfect and is the preferred method of choice (Whitley, 1994; Beasley, Bull, & Martin, 1993).

As discussed in Section 3.6.3, proportionate reproduction can have negative side effects. The selection pressure (chance of the best individual being selected) depends on the differences in fitness values in the population, and extremely fit individuals will cause the population to converge too quickly. The rest of the selection mechanisms discussed in Section 3.6.3 can be made to show perform similar performance by adjusting parameters, according to Goldberg and Deb (1991). We chose to implement linear ranking, which allows us to keep an even selection pressure throughout all generations of the GA. It also lets us manually adjust this pressure.

Selection probabilities in linear ranking selection are calculated as follows (Chakraborty & Chakraborty, 1997). The population is sorted on fitness value, with the worst individual receiving rank 1. Then, the probability of being selected is calculated for each individual as follows:

$$p_i = \frac{1}{N}\left(min + \frac{(max - min)(\mathrm{R} - 1)}{\mathrm{N} - 1}\right)$$

Where *N* is the population size, and *R* the rank of individual *i*. *Max* and *min* are two parameters used to adjust selection pressure, for which holds that *(max+min = 2)* and *(1≤max≤2).* A value of max closer to 2 means a higher selection pressure, whereas a value of 1 will give all ranks the same probability of being selected. The influence of the value of *max* on selection probability is shown in Figure 37 below, for a population size N of 10.

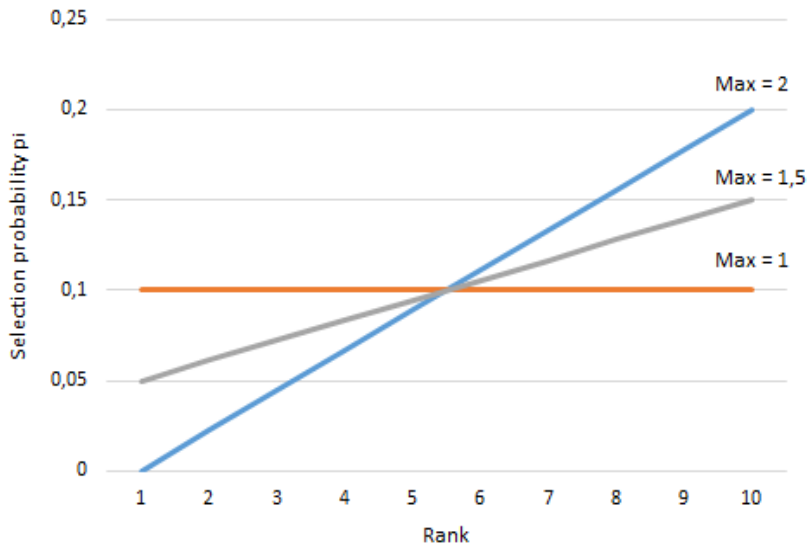


Figure 37. Selection probability per rank for linear ranking with different Max values

This approach has been chosen as it enables us to manually adjust the selection pressure, while still keeping the benefits of rank selection (see Section 3.6.3). Selection pressure is defined as the chance of the best offspring being selected into the next generation. As a final step, we investigate if putting one copy of the best solution/individual found so far into the new generation will improve the performance of the GA. We denote the variable that indicates whether or not this is done as *InsertBest*.

Parameters

Concluding, the following parameters can be changed for the SA algorithm:

- Number of generations (iterations).
- Population size.
- Value of *max* between 1 and 2 (adjustment of selection pressure).
- $P_{crossover}$ (chance of performing a crossover operation on any set of parents).
- $P_{mutation}$ (mutation rate, chance of any part of any offspring to become mutated, given by 1/notation string length).
- Classical or Elitist approach.
- *InsertBest* (Inserting the best solution found so far into the next generation or not).

4.6.4 Feasibility

When performing SA or GA on a partial schedule, multiple infeasible schedules will emerge. Especially when the partial schedule is already 'tight' in the sense that the floaters have little idle time in their schedules. To determine if a partial schedule is feasible, we have to determine if the floaters can visit each selected CSCs for the chosen duration of stay. In Figure 38, an example of a partial schedule (within the main schedule) is shown for one floater. When evaluating the feasibility of a partial schedule, the following criteria have to be met:

1. Each floater is able to reach the first CSC in the partial schedule in time, from the moment they finish their activities at the last CSC that is visited in the regular schedule before the partial schedule (indicated with '1' in Figure 38).
2. Within the partial schedule, a floater is able to reach each CSC from the partial schedule in time (indicated by '2' in Figure 38).
3. The floaters are able to reach the first CSC that follows the partial schedule, from the moment they are finished at the last CSC in the partial schedule (indicated with '3' in Figure 38).

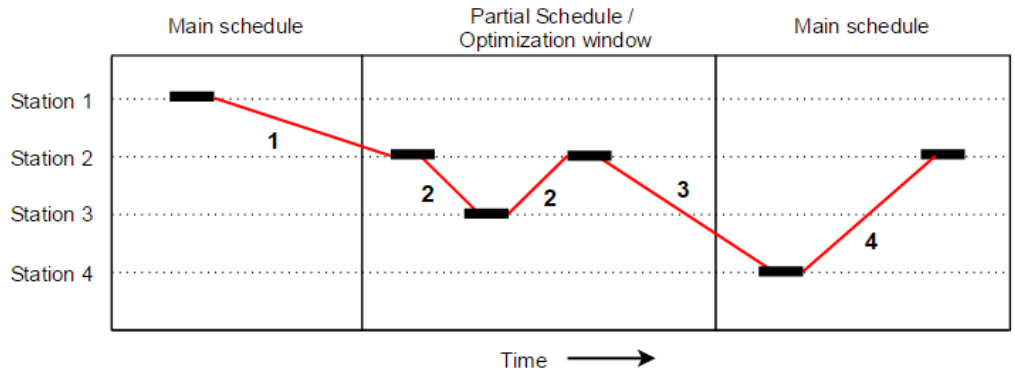


Figure 38. Feasibility check of partial schedules

The main schedule that is not changed in the optimization of the partial schedule, is always checked for feasibility before being accepted (indicated by '4' in Figure 38), so feasibility is guaranteed if the three criteria described above are met.

Both SA and GA use the objective function to select solution(s) on which to continue optimization. Infeasible solutions might contain information that is valuable to the search process. In our application, it could be possible that we have to 'step through' some infeasible solutions to get a better, feasible solution. In order to evaluate the objective function on an infeasible solution, we have implemented the following solution. If in the process of translating the notation string back to the partial schedule (see Section 4.6.1), one of the 3 feasibility criteria is violated, the CSC that caused this violation is deleted from the schedule. Therefore, any partial schedule that will be created in the optimization process, will be feasible (even though it might become an empty partial schedule).

4.7 Conclusion

The problem in this thesis is to construct a method that is able to produce floater schedules, with the goal of minimizing overtime and defects in the assembly line. In this chapter, we formulated a model based on the detailed problem description. To solve this model, we proposed a solution method to create schedules. An overview of this method and the different proposed elements, is shown in Figure 39.

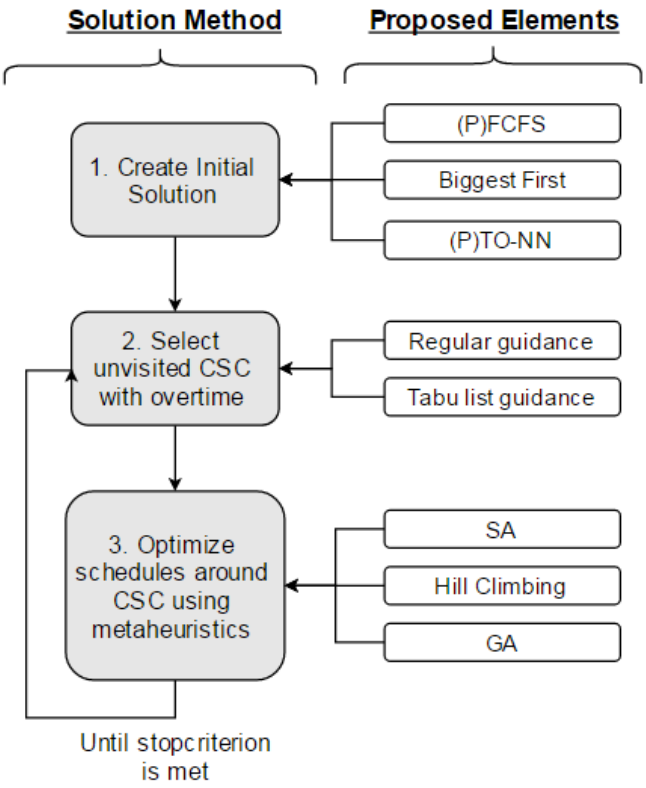


Figure 39. Proposed solution method and its elements

This method first uses a constructive heuristic to create an initial schedule (see Section 4.5). This heuristic uses one of 3 proposed selection and addition rules discussed in Section 4.5.2.

Next, the initial schedule is optimized by iteratively selecting CSCs on which overtime occurs, according to guidance rules of which we proposed 2 variants in Section 4.6.1. The floater schedules around each selected CSC (called partial schedules) is then optimized. For the optimization of partial schedules, we propose 2 metaheuristics, Simulated Annealing (Section 4.6.2) and a Genetic Algorithm (Section 4.6.3). Hill Climbing will be used as performance reference.

5. Numerical Analysis

In this chapter, we optimize the parameters of our solution method based on the configuration of the assembly line at Company A and the provided dataset. Our solution consists of multiple steps, and for each step multiple methods have been proposed. We optimize the parameters for each method, and then select the best method to perform each step. First we discuss how the solution methods have been implemented in Section 5.1. The experimental factors are then discussed and an experimental plan is constructed in Section 5.2 and Section 5.3. We start optimizing the parameters for the initial solution methods in Section 5.4, followed by the metaheuristics in Section 5.5, and the Guidance Rules in Section 5.6.

5.1 Implementation

As discussed in Chapter 1, the solution method developed in this thesis has been implemented into the Planning System that was developed by Ergo-Design B.V. in the software package 'Plant Simulation'. In Appendix B, a screenshot of the Floater Scheduling frame in Plant Simulation can be found.

Plant Simulation primarily uses tables to store data, and has been optimized to perform quick calculations on these tables. This makes the program suitable to store, retrieve and alter the data discussed in Section 4.3.3 in the form of a CSC list and a Floater Schedule as shown in Figure 24 in Section 4.3.3. The locations of the stations, number of workers per station, the station sequence, and the processing times (of different car options on the stations) are embedded in the current model. This information is loaded from a configuration file. Next, a dataset is loaded into the system, which contains the cars to be produced and their configuration (and resulting processing times) for a specific day and shift. The sequencing module of the planning system constructs a sequence for these cars. This sequence is then used as input for the Floater Scheduling module, which uses the methods developed in this thesis.

Next to the methods described in Chapter 4, auxiliary scripts were implemented to perform the following tasks:

- (Re)Calculating the starting/finishing times, delay and overtime for all CSCs when schedules are changed. A change in the schedules consists of a change in the duration of stay R at a CSC. Starting from this CSC, the quantities are recalculated for every succeeding CSC, until the finishing time of the CSC is the same for the new and the old situation. This process prevents us from having to recalculate the overtime on all CSCs when evaluating the objective function for a solution, saving CPU time.
- Checking the feasibility of the floater schedules (see Section 4.6.4).
- Visualizing the floater schedules in Plant Simulation.

Line configuration

As discussed in Section 4.1, the floater scheduling model should be adaptable to different situations. To achieve this, the parameters in the model have been kept variable. The following settings can be changed to match different line configurations:

1. Overlaps of stations (Section 4.2.1).
2. Travel and setup times (Section 4.2.8).
3. Floater efficiency (Section 4.3.3).
4. Division of the total assembly line into working areas (Section 4.4.1).
5. Number of available floaters per area (Section 4.4.1).

For experimental purposes we set the configuration parameters listed above at the following values for the remainder of this thesis. Overlaps (both negative and positive) of the stations are fixed at 5 seconds for all stations. Floaters travel with 5 km/h, and a setup time of 15 seconds at each station is used. Floaters are assumed to work as fast as regular workers (floater efficiency). The division of the assembly line into working areas is identical to the division into lines. The number of floaters per area is determined in Section 6.2.

5.2 Experimental factors

In Section 4.4.5 we proposed a solution method. The main part of this method is shown again on the left in Figure 40. On the right, we have added the different parameters and design choices that might affect the performance of our solution method. These parameters will be our experimental factors.

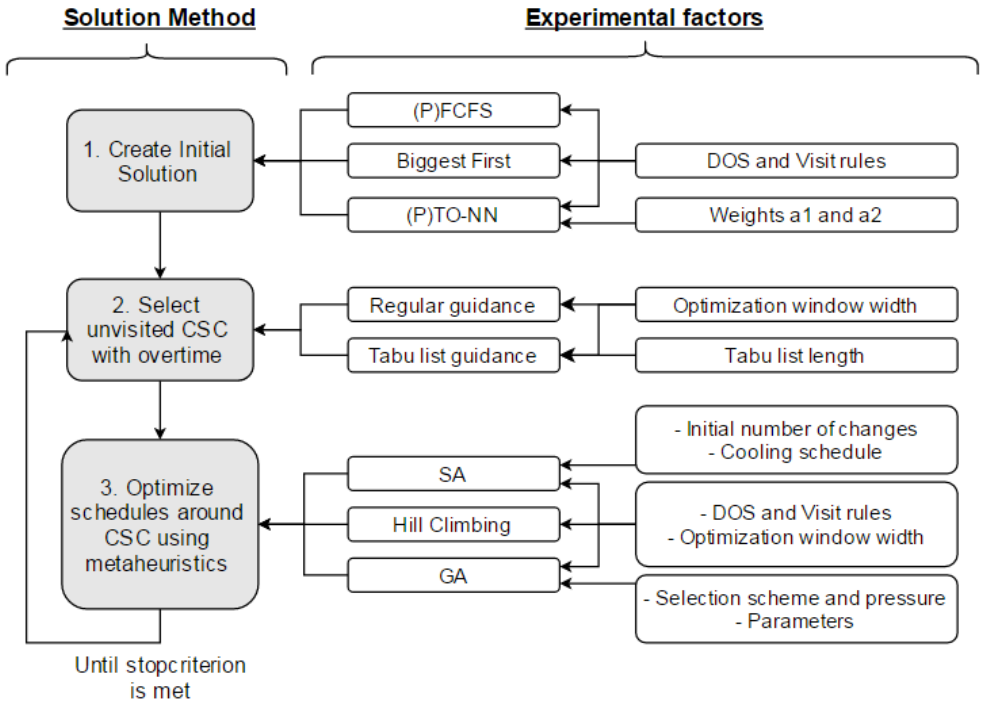


Figure 40. Experimental factors

5.3 Experiment plan

As it is impossible to test every combination of factor settings, we create an experiment plan based on our insight in the problem combined with common sense.

5.3.1 Initial Solution

A starting solution is needed in order to use the improvement methods. Therefore we first analyze the different constructive heuristics (FCFS, Parallel FCFS, TO-NN, and Biggest First) in Section 5.4 to see which one performs best.

5.3.2 Metaheuristics for partial schedules

One way to look at our solution method, is that it resembles a local search heuristic (optimization of partial schedules at step 3 in Figure 40) within another local search heuristic (selection of CSCs at step 2). The guidance rules at step 2 of the process iteratively calls step 3, which then returns an improved schedule.

Optimizing step 3 can be done independently of the higher-level Guidance rules at step 2. The Guidance rules do not influence the search space for the metaheuristics that will optimize the partial schedule. After the Guidance rule has selected a partial schedule, the best performing metaheuristic should be used. Therefore, we first analyze and optimize the metaheuristics proposed for performing step 3 in Section 5.5.

The metaheuristics discussed in this section are iteratively called by the higher level guidance rules (which will be discussed in Section 5.6). At each call, the metaheuristic will perform a local search. Because the performance of a metaheuristic at a single call does not give legitimate insight, we use a basic framework in which each metaheuristic will be called a fixed number of times. This framework is as follows:

- We retrieve a starting solution by using the best initial solution method.
- We then sort all candidate CSCs on overtime in descending order.
- Starting from the top, we select a candidate CSC and iteratively call the metaheuristic under examination (to optimize the partial schedule around that CSC)
- We do not sort the candidate CSCs between selections of CSCs.
- A fixed number of 100 partial schedules will be optimized with each metaheuristic.

This ensures that each metaheuristic will be executed a multitude of times. For each configuration of metaheuristics, we then report on the **average reduction in objective function per partial schedule**. The objective function used is described in Section 4.3.2. Results can be compared as each metaheuristic will be presented the same partial schedules to optimize.

5.3.3 Guidance rules for selecting CSCs

Next, with the best performing methods selected for step 1 and 3, we perform experiments on the different guidance rules for step 2. In Section 5.6 we will search for the best balance between optimizing more partial schedules and the quality of each partial schedule.

5.3.4 Run time

Because floater schedules have to be created each day, the available CPU time is limited. The method should produce schedules in approximately 30 minutes as mentioned in Section 4.1. We might have to find a balance between the number of partial schedules that we can optimize, and the quality of each partial schedule. Spending more time optimizing each partial schedule can result in better partial schedules, but as a result less partial schedules can be optimized. We further discuss how we tackle this challenge in Section 5.6.

All methods were tested on a laptop running Window 8 on an Intel i7 processor at 2.9 GHz, with 8 gigabyte of RAM memory. The 32 bits version of Tecnomatix Plant Simulation 9 was used, which therefore only utilizes 4 gigabyte of RAM.

5.3.5 Current way of working

We reconstruct the current way of working using a constructive heuristic. In the current situation, overtime is not anticipated, and floaters are reactively sent to provide help. This is reconstructed by letting a station request a floater, at the moment a car which will cause overtime enters the station. A floater will then travel to that station (if one is available) and stay there until the station starts its activities at the start of the negative overlap. After that, the floater waits until he gets a new call for help from another station. If no such call is received by the floater within 30 seconds, he moves to the center of the line and waits there for the next call.

5.3.6 Datasets and floaters per area

For testing purposes, datasets are available for 10 workdays, consisting of two 8 hour shifts. These datasets include the cars to be produced and their configuration. The planning system then creates the sequence of cars that will be used as input for our floater scheduling methods. There is no information available on how many floaters are available for each area. For testing purposes, we assign floaters to each area depending on area characteristics. In Table 8, we report average data of all working areas, over the available datasets. The average amount of overtime, number of defects, and resulting objective function (O.F.) are shown. We assigned floaters to each area as follows. Initially, we assigned 1 floater to each area, as all areas show some degree of overtime. Next, we applied the heuristic that simulates the current way of working on each area, and the resulting average (reduction of) objective function is shown in the bottom two rows of Table 8.

Area	1	2	3	4	5	6	7	8	9	10	11
Avg. Overtime	95,3	2323,2	4774,6	18829,5	2727,1	4429,8	13424,2	562,9	41,9	404,5	81,0
Avg. Defects	35,0	73,0	402,6	1273,4	96,4	300,8	910,2	69,6	7,0	67,2	26,8
Avg. O.F., No Floaters	444,8	3052,8	8800,2	31563,0	3690,4	7437,2	22525,8	1258,4	111,8	1076,0	348,4
Avg. O.F., Current Situation, 1 floater	0,0	748,7	1553,1	16098,5	535,8	399,1	18393,2	0,0	0,0	9,2	0,0
Avg. Reduction	100%	75%	82%	49%	85%	95%	18%	100%	100%	99%	100%

Table 8. Working Area characteristics

We have assigned 2 floaters to those areas for which the current way of working with one floater resolves less than 90% of the objective function. This considers Areas 2, 3, 4, 5, and 7.

The problem subdivision discussed in Section 4.4.1 allows us to view each area as individual problem. We assume that the method that shows the best performance on the most difficult area(s), will also show the best performance in other areas.

Therefore we select the most problematic area to perform our analysis. From Table 8, Areas 7 and 4 seem to be the most problematic. Area 7 has less overtime and defects than Area 4 on average, but a much smaller percentage of these problems is resolved by applying 1 floater and using the current way of working. The stations in Area 7 are further apart (see Figure 41), which increases the complicating factor of traveling distances in determining floater schedules. Therefore we have selected both Area 7 and Area 4 for our numerical analysis.

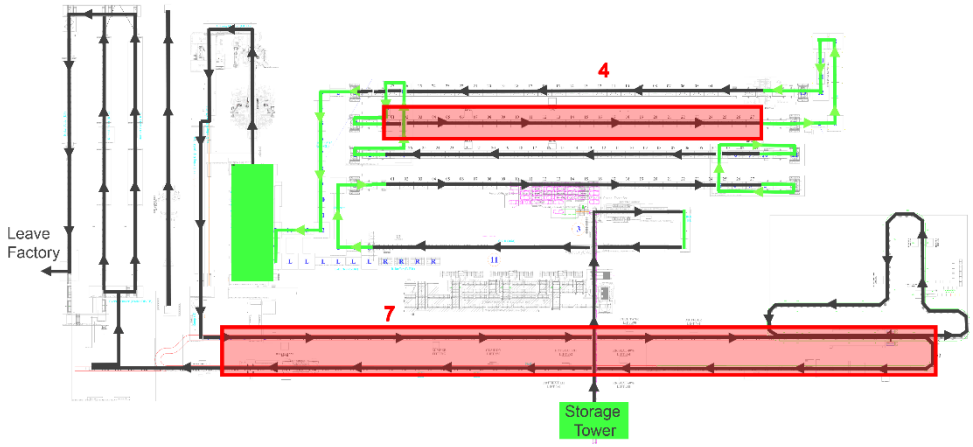


Figure 41. Working Areas under consideration

In the available datasets, Area 7 contains a total of 106 stations, and on average 29 of those stations have overtime during a shift. The maximum distance between any two of these stations is 250 meters. Area 4 contains 30 stations, from which 18 have overtime on average, with a maximum distance between two of these stations of 150 meters.

5.4 Optimization of initial solution methods

The constructive heuristics to create initial solutions are proposed in Section 4.5 and shown in Figure 40. In Table 9, we show the average (over all datasets) reduction in objective function (as percentage of the current way of working) for every combination of Duration of Stay rule, Visit rule and Constructive Heuristic for Areas 4 and 7. The average reduction per DOS and Visit rule is shown between brackets.

Area 4			
DOS	Visit	Heuristic	Reduction
1 [62,1%]	1 [61,3%]	FCFS	61,2%
		PFCFS	63,5%
		BiggestF	59,2%
	2 [62,9%]	FCFS	64,4%
		PFCFS	65,1%
		BiggestF	59,2%
2 [57,0%]	1 [57,9%]	FCFS	58,3%
		PFCFS	59,5%
		BiggestF	55,9%
	2 [56,1%]	FCFS	55,7%
		PFCFS	56,7%
		BiggestF	55,9%
3 [52,2%]	1 [51,1%]	FCFS	49,1%
		PFCFS	48,6%
		BiggestF	55,5%
	2 [53,3%]	FCFS	52,9%
		PFCFS	51,4%
		BiggestF	55,5%

Area 7			
DOS	Visit	Heuristic	Reduction
1 [79,3%	1 [79,4%]	FCFS	77,9%
		PFCFS	81,6%
		BiggestF	78,7%
	2 [79,2%]	FCFS	78,3%
		PFCFS	80,5%
		BiggestF	78,7%
2 [79,1%	1 [79,0%]	FCFS	77,2%
		PFCFS	81,0%
		BiggestF	78,8%
	2 [79,1%]	FCFS	78,5%
		PFCFS	80,1%
		BiggestF	78,8%
3 [78,6%	1 [78,3%]	FCFS	76,2%
		PFCFS	80,1%
		BiggestF	78,7%
	2 [78,8%]	FCFS	78,1%
		PFCFS	79,6%
		BiggestF	78,7%

Table 9. Performance of initial solution methods

From these numbers, we draw the following conclusions:

- Duration of stay rules outperform each other on average in the order of *Rule 1> Rule 2> Rule 3*.
- There is no visit rule that systematically outperforms the other.
- Parallel First Come First Serve seems to outperform other heuristics, as it gives the best results for 10 of the 12 combinations of DOS and Visit rules (only outperformed on Area 4 for DOS rule 3).
- Overall, the combination of Parallel First Come First Serve and DOS rule 1 shows good performance.

We believe that the reason DOS rule 1 outperforms the others, is because of the way the constructive heuristics work. The constructive heuristics iteratively try to add a CSC to the floater schedules. The only CSCs that are considered are those with overtime themselves. Consider this example: CSC #1 has no overtime, but delays the start of CSC #2. The constructive heuristic will only consider visiting CSC #2. DOS rules 2 and 3 try to let the floater stay longer than is needed to resolve the overtime (with the goal of decreasing the start time of the next CSC). If there is no CSC with overtime following CSC 2, this extra time is wasted. Apparently, CSCs with overtime are not often enough followed by another CSC with overtime for this approach to be beneficial. In our example, visiting CSC #1 with DOS rule 2 or 3 could have helped, but CSC #1 is not considered by the constructive heuristics. Therefore, DOS rule 1 shows the best performance.

We select Parallel First Come First Serve, DOS rule 1, and Visit rule 2 for creating initial solutions (schedules) to start our optimization. This setting shows the highest average reduction in objective function over both areas (72,8%). Note that during optimization, a different configuration of DOS and Visit rules can be used.

Omitting TO-NN

The only method with parameters proposed by us is the Time Oriented Nearest Neighbor method (see Section 4.5.2). In our analysis, this method showed the worst performance of the three proposed methods, regardless of parameter settings. We expect this to be the results of CSCs being selected too far in the future. The TO-NN will sometimes select a CSC that is further away in the future, when it has a large amount of overtime, even though it would have been possible to visit other CSCs first. If a CSC that occurs 5 takts into the future is selected for a floater, he will not provide assistance at any CSCs in the meantime. The best results for TO-NN were retrieved with the weight values that made the fraction a1/a2 very large. This pushes the TO-NN heuristic towards a FCFS principle, but the performance was still considerably worse than both FCFS and Biggest First, as shown in Table 10 (Based on one shift, using DOS rule 1 and Visit rule 2).

	Area 4		Area 7	
	Objective value	CPU Time (s)	Objective value	CPU Time (s)
FCFS	4119.44	0.47	5911.09	0.82
PFCFS	4100.10	0.54	5451.55	0.84
BiggestFirst	4996.53	0.48	5623.50	0.81
TO-NN	8440.65	30.79	9185.49	12.76
PTO-NN	12876.94	24.24	9863.35	12.05

Table 10. Example of realized objective values and used CPU time per initial solution method

The running time was also considerably larger than that of the other two heuristics. TO-NN showed an average running time of around 30 seconds, compared to the less than 1 second for FCFS and Biggest First. This is caused by the fact that in each iteration, TO-NN calculates scores for a number of CSCs into the future (20 in our case) and has to sort all candidate CSCs according on these scores. For the reasons discussed above, we have omitted TO-NN from our results.

5.5 Optimization of metaheuristics

In this section, we analyze the effects of different parameters for each metaheuristic used in step 3 of the solution method from Figure 40. We optimize each heuristic individually, and finally compare their performance to select the best option to be used in our solution method.

We analyze the performance of the metaheuristics based on 1 dataset (shift). The datasets do not differ from each other significantly, as the same models of cars (in approximately the same proportions) are produced in each shift. Also, the configuration of the assembly line itself (positions of stations, number of workers, etc.) does not change between shifts. Using one dataset allows us to perform a more detailed analysis of parameters.

SA, GA and HC all consider a number of neighbors, and for each of these neighbors the objective function is evaluated. The total number of **Evaluated Neighbor Solutions (ENS)** is given by:

- SA: (# iterations) * (# of neighbors considered at each iteration)
- GA: (# generations) * (population size)
- HC: (# iterations)

For each metaheuristic, evaluating the objective function takes up the majority of the running time. We add ENS as experimental factor and aim to find the configuration that most efficiently searches an assigned number of ENS for each metaheuristic. When we have found such a configuration, we can vary the number of ENS during the optimization of the higher-level guidance rules. The number of ENS allows us to indicate how much effort should be put into optimizing a single partial schedule, and enables us to compare the performance of the different heuristics. From preliminary tests, evaluating a neighbor solution takes around 12 milliseconds. For our experiments, we have selected 500, 1000, and 2000 as the different levels for the factor ENS, resulting

in a predicted time per partial schedule of 0.6, 1.2 and 2.4 seconds.

Parameters independent of metaheuristic

There are three parameters that play a role in the optimization of partial schedules for all the proposed metaheuristics. The DOS and Visit rules, and the optimization window width. The DOS and Visit rules do not influence the way metaheuristics search the solution space. These rules translate the decisions made by the metaheuristics (which CSCs to visit) into floater schedules. They do not directly influence the guidance of the search process. If one configuration outperforms another, we expect this to be true for any setting of DOS and Visit rules. Initially, we choose a setting of DOS rule 2 and Visit rule 1 for optimizing the parameters of the metaheuristics. This is a different combination of rules as used for the initial solution. This is done because DOS rule 2 will allow us to visit a CSC even when the CSC itself does not have overtime, which gives more options to resolve overtime.

When a CSC is selected by the guidance rules, the selected metaheuristic will optimize the floater schedules in a time window around that CSC, that time window is called the optimization window. The width of this window influences the search space, a wider window means there will be more candidate CSCs that can be visited by a floater. We initially use an optimization window width of 5 minutes for all metaheuristics, containing 21 candidate CSCs on an average for Working Area 4, and 33 for Working Area 7 (which is expected as Working Area 7 contains more stations). We will test combinations of ENS, Guidance Rules and Optimization Window widths in Section 5.6.

Starting solution and performance reference

We retrieve a starting solution using Parallel First Come First Serve combined with DOS rule 1 and Visit rule 2. As a reference for the performance of our metaheuristics, we use the HC algorithm. HC creates neighbor solutions and only accepts one if it improves the objective value. We implemented HC using the same neighbor structure as SA. We tested HC on 100 partial schedules using the same framework as for SA and GA, described in Section 5.3.2. The average reduction in objective function per partial schedule for 500, 1000 and 2000 ENS are shown in Table 11 (Data can be found in Appendix C).

ENS	Area 4	Area 7
500	14,40	14,68
1000	14,78	15,18
2000	15,11	15,62

Table 11. Performance of the Hill Climbing algorithm

5.5.1 Simulated Annealing

For SA, the list of parameters that can be changed is as follows:

- Number of iterations *I* (number of temperature changes)
- Number of neighbors *N* created at each iteration (at each temperature).
- Starting temperature (T_{start}).
- Temperature function (option A or B in Figure 34).
- Cooling rate θ (for cooling schedule B)
- Initial number of changes (the initial number of swaps/moves per neighbor).

Starting temperature

We proposed two temperature functions in Section 4.6.2. The starting temperature T_{start} is problem specific, and once a suitable starting temperature is chosen, it can be used in any configuration of parameters. As discussed in Section 4.6.2, the probability that a solution is accepted that deteriorates the objective function, is given by $e^{-\delta/Temperature}$. SA should start at a temperature T_{start} such that initially, virtually all proposed neighbor solutions will be accepted.

At each temperature, a number of N neighbors is considered. We denote the number of neighbors that is accepted as n*. The acceptance ratio is then defined as n*/N, and should be close to 1 at the start of the SA process (for the first couple of iterations). To find a suitable T_{start}, we monitored the acceptance rate during SA for different starting temperatures. A starting temperature of 300 was selected. Figure 42 shows the acceptance ratio at each iteration for T_{start} = 300, for both the exponential and linear cooling schedules.

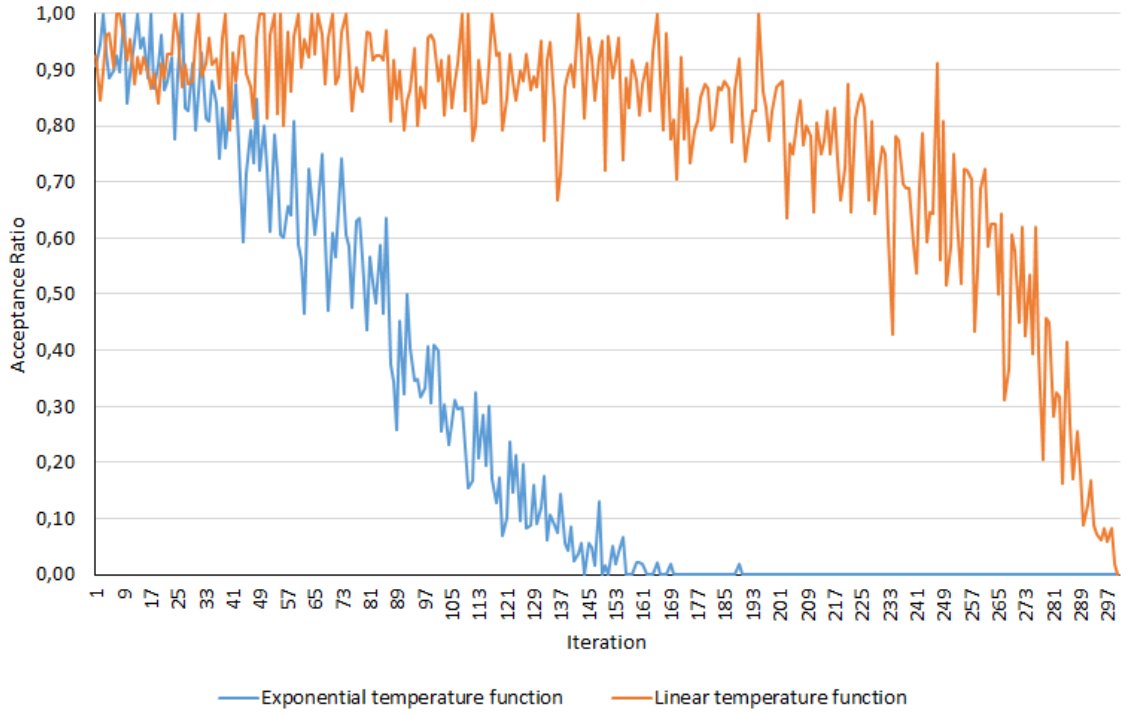


Figure 42. Acceptance ratio vs iterations for a single run of SA (one partial schedule), for T_{start}=300

Cooling rate

A larger number of iterations for SA gives a larger number of temperature changes. We vary the number iterations/temperature changes in our analysis. This means that configurations with a smaller number of iterations, we have to cool more quickly (larger decrease per temperature change) in order to approach 0 at the end of the SA process. To achieve a consistent shape of the temperature curve over the iterations, the cooling factor θ should be adjusted if temperature function B is used (see Section 4.6.2). For the linear temperature function A, the temperature function is already consistently defined for different numbers of iterations.

We devised a formula for the value of θ by stating that after a fraction f of the total number of iterations *i* has been completed, the starting temperature T_{start} should have dropped to a low temperature of T_{goal}. The formula for the cooling factor θ is:

$$\theta = \frac{T_{goal}}{T_{start}}^{1/(f*i)}$$

This formula allows us to adjust the cooling parameter according to the number of iterations and starting temperature. We have chosen f=0.5, and T_{goal}=5. This results a temperature function in which the temperature is relatively low for the last 50% of the iterations, to allow the SA algorithm to solidify around an optimum. For example, for T_{start}= 300 and 500 iterations, the formula gives a cooling factor θ of 0,983.

Optimization of parameters

The DOS and Visit rules, together with the optimization window width, have been set at a fixed value in the introduction of Section 5.5. We have defined the parameters of our two temperature functions, and left i_{freeze} out of consideration for now. This leaves us with the following parameters:

1. Number of iterations *i* (number of temperature changes)
2. Number of neighbors *N* created at each iteration (at each temperature).
3. Temperature function (Linear or Exponential).
4. *Initial number of changes* C (the initial number of swaps/moves per neighbor, which will decrease to 1 over the total number of iterations).

For the number of *neighbors* evaluated at each iteration, we have defined 4 levels: 2, 5, 10, and 20. We defined three levels of ENS (500, 1000 and 2000), which is the total number of evaluated neighbor solutions. Therefore, the number of iterations is not an experimental factor and is given by calculating (ENS/N). For each configuration we test the two temperature functions (linear function A and exponential function B) and two levels for the *initial number of changes* (1 and 3), to find out if more than one initial change is preferable for certain configurations (1 and 3). This finalizes our experiment settings, and gives a total number of 4 factors and 48 different configurations (16 per number of ENS, see Table 12).

ENS	*Neighbors* created at each iteration	Number of iterations i	Temperature function (A=linear, B=exponential)	Initial number of changes (1 or 3)
500, 1000, 2000	2, 5, 10, 20	(ENS/i)	A or B	1 or 3

Table 12. Experimental settings for SA

Using the framework discussed in Section 5.3.2, each configuration is applied to optimize the same 100 partial schedules. We then store the average reduction in objective function per partial schedule, *AverageReduction*. This is done for both Area 4 and Area 7, giving two sets of data. We use this data to find the best configuration of SA for each number of ENS that we assign.

Results

The results for Simulated Annealing can be found in Appendix D and E. Each neighbor evaluation took around 1.3 milliseconds, which is almost entirely used for evaluating the objective function. The CPU times per partial schedule for 500, 1000, and 2000 ENS are 0.69, 1.35, and 2.66 seconds respectively.

As expected, a configuration with a larger number of ENS resulted in better partial schedules overall. In Table 13, the average reduction in objective function per partial schedule is shown per number of ENS and Area. For each number of ENS, the reduction is averaged over all configurations using that number of ENS.

		Area 4	Area 7
ENS	2000 ENS	22,43	22,05
	1000 ENS	19,52	21,22
	500 ENS	16,88	19,86

Table 13. Average reduction in objective function using SA, per ENS and Working Area

The best performing configurations for each number of ENS are shown in Table 14 and 15 for Working Areas 4 and 7.

ENS	Iterations	Neighbors	TempFunction	NrChanges	A.R.
2000	200	10	2	1	28,20
1000	200	5	2	1	26,36
500	250	2	2	3	24,89

Table 14. Best performing SA configurations on Area 4

ENS	Iterations	Neighbors	TempFunction	NrChanges	A.R.
2000	100	20	2	3	22,50
1000	100	10	2	3	22,23
500	50	10	2	3	22,22

Table 15. Best performing SA configurations on Area 7

We want to select the best configuration for each number of ENS that shows the best performance on Areas 4 and 7. Therefore, we rank the performance of each configuration on Average Reduction (A.R.) for each number of ENS separately. We do this for both Working Areas. The configuration that has the highest average rank (over Area 4 and 7) for its number of ENS is then selected. The selected configurations are shown in Table 16.

ENS	Iterations	Neighbors	TempFunction	NrChanges	Avg. Rank (For ENS)	A.R. Area 4	A.R. Area 7
2000	100	20	2	3	1,5	28,09	22,50
1000	200	5	2	1	1,5	26,36	22,21
500	50	10	2	3	1,5	24,73	22,22

Table 16. Selected SA configurations with highest average performance ranking

We now further examine the influence of the different parameter settings on performance. A multiple regression was run to predict *AverageIncrease* from *Neighbors, Temperature function, NrChanges* and *ENS*. This was done for each Working Area independently. Each configuration resembles a data point (n = 48). Note that each configuration was run on 100 partial schedules.

Both regressions statistically significantly predicted AverageIncrease. The results can be found in Appendix F, and show that (next to the number of ENS) the temperature function is the only parameter that significantly

predicts the performance of a configuration (for a p-value < 0.000) for both Areas. The exponential temperature function (function B) performs better than the linear temperature function (function A), as shown in Table 17.

		Area 4		Area 7	
		T_Linear	T_Exponential	T_Linear	T_Exponential
ENS	500 ENS	10,17	23,58	18,40	21,32
	1000 ENS	13,83	25,20	20,51	21,93
	2000 ENS	17,78	27,08	21,75	22,36

Table 17. Average reduction in objective for every combination of Area, ENS, and Temperature function

The number of neighbors was not a significant predictor of Average Reduction for both Area 4 and 7 (p-values of .811 and .177 respectively). The number of *Neighbors* influences the amount of neighbors that are evaluated at each temperature. For example, if a temperature function is used in which the temperature drops very rapidly, increasing the number Neighbors would mean that more solutions are considered at a 'high' temperature. For the temperature functions used by us, the number of neighbors at each temperature does not significantly influence the performance of SA.

The number of initial changes was close to being significant for Area 4, and was significant for $p<0.05$ for Area 7 with a p-value of 0.025. The average reduction in objective function for each combination of *ENS*, *Area* and *NrChange*s is given in Table 18.

		Area 4		Area 7	
		1 Change	3 Changes	1 Change	3 Changes
ENS	500 ENS	16,33	17,42	19,42	20,31
	1000 ENS	19,02	20,02	20,99	21,44
	2000 ENS	22,47	22,40	21,86	22,25

Table 18. Average reduction in objective function per partial schedule, for 1 and 3 initial changes

For both areas, a lower number of ENS (500 and 1000) performs better on average combined with a higher number of Initial Changes. We suspect this is caused by how the solution space is searched by the SA heuristic. The complete solution space is made up by all possible neighbor solutions. Every time a neighbor is created, the search process makes a step in a certain direction in this space. A low number of ENS means that the search process has less 'steps' to cover the complete solution space. A higher number of Initial Changes means that each step is larger (in the beginning), allowing for a better coverage of the solution space, which in turn results in a better performance. Configurations with 2000 ENS have more steps to cover the solution space, and have a smaller need for an increased step size in case of Area 4. Here, increasing the number of Initial Changes (step size) results in a worse performance as the search process steps over potentially good solutions (for Area 4). Area 7 contains more stations, resulting in a larger solution space and a higher number of initial changes again performs better for 2000 ENS. Overall, when the number of ENS is small in proportion to the solution space size, a higher number of initial changes performs better on average.

Conclusion Simulated Annealing

In this section, we have set the starting temperature of SA at 300, and provided a formula to calculate the cooling factor for each number of iterations. Configurations that use the exponential temperature function perform significantly better than those using the linear function. The number of neighbors at each iteration/temperature did not significantly influence the performance of SA. On average, a higher number of Initial Changes showed better performance when the number of ENS was small compared to size of the solution space. In Table 19 the selected configuration for SA is shown for each number of ENS.

ENS	Iterations	Neighbors	TempFunction	NrChanges
2000	100	20	2	3
1000	200	5	2	1
500	50	10	2	3

Table 19. Selected configurations for SA for each number of ENS

5.5.2 Genetic Algorithm

For the optimization of the parameters for GA we refer to appendix G. The approach used for SA was also used for GA. In our analysis GA showed poor performance. GA always performed worse than SA, and in some cases even worse than our reference algorithm HC.

We conclude that the implemented Genetic Algorithm is not a suitable choice for optimizing the partial schedules in our application. This might result from the fact that the Genetic Algorithm is less likely to perform multiple mutations on the same solution. Simulated Annealing (and the Hill Climbing algorithm) make ‘a lot of changes to a single solution’ in every iteration, whereas the GA makes ‘a few changes to a lot of solutions’. The process of GAs should be designed such that these changes can be preserved (diversity in the population), so new and better solutions can be found by recombining and mutating individuals multiple times. But despite our efforts to introduce and maintain diversity in the populations, the populations converged around sub optimal solutions, and GA was unable to efficiently search the solution space in our application.

5.5.3 Selection of the best metaheuristic and final settings

In Section 5.5.1 and 5.5.2 we performed a numerical analysis on the metaheuristics (SA & GA) that are used to optimize partial schedules. A summary of the results is shown in Table 20, together with the reference algorithm, HC. SA* and GA* indicate the best performing configurations on each Working Area. SA indicates the selected configurations that showed the best average performance over Areas 4 and 7.

	Area 4				Area 7			
ENS	**SA**	**SA***	**GA***	**HC**	**SA**	**SA***	**GA***	**HC**
2000 ENS	28,09	28,20	23,23	14,57	22,50	22,50	13,95	14,68
1000 ENS	26,36	26,36	20,19	14,86	22,21	22,23	13,67	15,18
500 ENS	24,73	24,89	17,10	14,63	22,22	22,22	13,54	15,62

Table 20. Average reduction in objective function per partial schedule, for each metaheuristics and number of ENS

We have selected Simulated Annealing for optimizing partial schedules, as this algorithm showed the best performance for each Area and number of ENS. For Area 4, the second best performing algorithm is the Genetic Algorithm, which on average realized 76% of the reductions found by SA. On Area 7, the gap between SA and the second best algorithm is even larger, with HC only reaching 69% of the reductions found by SA. To give a side by side comparison, we let each algorithm optimize the same 100 partial schedules (as discussed in Section 5.5) for Area 7. In Figure 43, the objective function is shown as a function of the number of partial schedules optimized. For SA, the selected configuration that showed the best average performance over Area 4 and 7 is used. For GA, the best performing configuration for Area 7 is used.

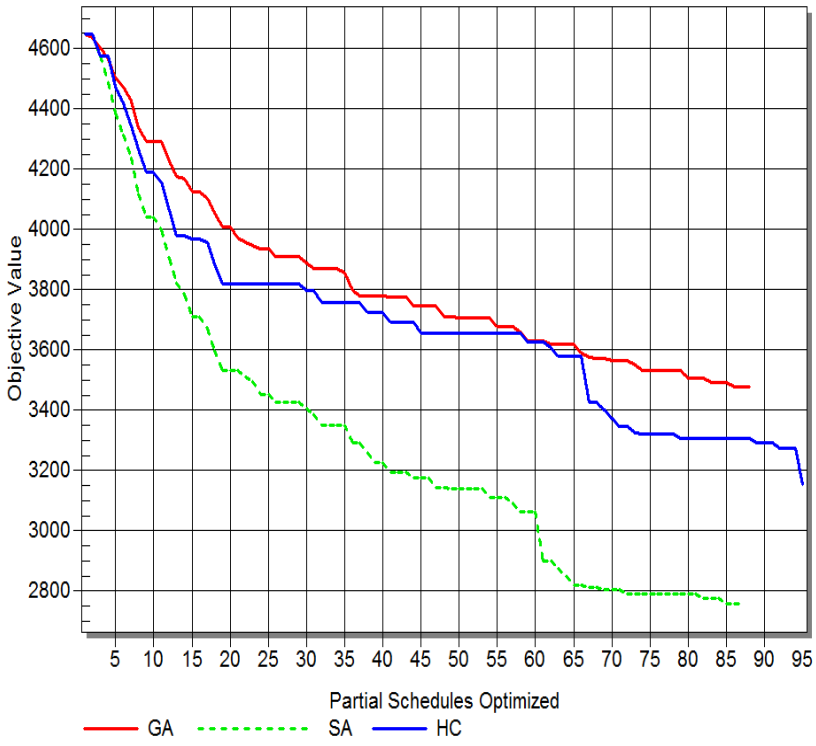


Figure 43. Comparison of metaheuristics on Area 7

The lines end at different points on the x-axis, because when a CSC has no overtime by the time it is selected, the algorithm will not optimize the partial schedule around it. From now on, we use SA for optimizing partial schedules. For each number of ENS, we use the configuration that was selected in Section 5.5.1.

Duration Of Stay and Visit rules

Now that we have selected a metaheuristic, we will look at the final parameters: DOS and Visit Rules and the optimization window width. We argue that DOS and Visit rules can be set independently of window width or the number of ENS that is chosen, as they do not change the size of the solution space, only the associated objective functions. Therefore, we test each combination of DOS and Visit rules for 1000 ENS on Area 4 and 7 and an optimization window width of 5 minutes. Results are shown in Table 21.

DOS	Visit	Area 4	Area 7	Average
1	1	29,15	23,48	26,32
1	2	30,26	7,36	18,81
2	1	23,82	21,74	22,78
2	2	18,92	6,20	12,56
3	1	24,69	22,34	23,51
3	2	7,59	5,10	6,35

Table 21. Performance of DOS and Visit rules for Area 4 and 7

The most remarkable result is the bad performance of visit rule 2, especially on Area 7. The duration of stay rules prescribe a duration of stay for each CSC in a schedule. The Visit rule determines what happens if a floater cannot stay at that CSC for this prescribed amount of time. Visit rule 2 skips the CSC altogether if the prescribed duration of stay cannot be met. We suspect that this causes too many CSCs to be skipped, missing opportunities to resolve overtime. This effect is increased as a longer duration of stay is prescribed, with the worst results being achieved by DOS rule 3.

We select DOS rule 1 and Visit rule 1, as this combination shows the best performance.

Influence of objective function on performance of DOS/Visit rules

We note that the performance of DOS and Visit rules is related to the objective function. Because any amount of overtime on a CSC is considered a defect, visiting that CSC for a time shorter than the overtime (not resolving all overtime) does not improve the part of the objective function associated with defects. When more emphasis is put on defects, by increasing the weight of defects in the objective function, we expect an increasing performance of Visit rule 2. We illustrate this by plotting the relative performance of Visit rule 2 compared to rule 1 in Figure 44. DOS rule 1 was used with 1000 ENS using the best SA configuration on Area 4. On the horizontal axis, the weight of defects in the objective function is shown (the weight for overtime is fixed at 1).

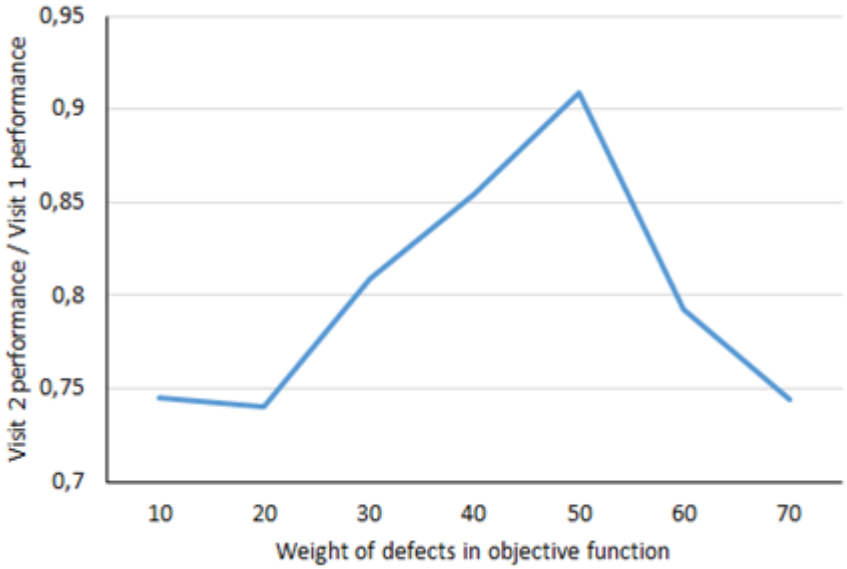


Figure 44. Relative performance of Visit rule 2 to rule 1

As more emphasis is put on defects in the objective function, the relative performance of Visit rule 2 compared to rule 1 increases. In our application there is a turning point at a weight of 50, where no additional defects can be resolved, and visit rule 2 starts to perform worse. However, Visit rule 2 does not surpass Visit rule 1 in performance.

Optimization window width

We attempted to determine an optimal optimization window width using the testing framework discussed in Section 5.3.2. We tested a window width starting from 3 minutes up to a window of 11 minutes, for each number of ENS. Results are shown in Table 22, and are averaged over ENS.

Window	Average Reduction
3	24,46
5	24,68
7	29,11
9	33,11
11	34,64

Table 22. Average reduction per window width

A larger window width resulted in a higher average reduction per partial schedule. However, this result is biased. Using the testing framework (discussed in Section 5.3.2), we run the algorithm on the same 100 CSCs. A wider optimization window may include multiple CSCs with overtime, and may also include CSCs that are not included in the original 100 CSCs. Instead of changing the schedules around one CSC in an attempt to resolve its overtime, the algorithm now also resolves overtime on multiple CSCs at the same time. This is not a problem on its own, but a narrow optimization window results in a more extensive search of the solution space as it is smaller. The idea behind optimizing small parts of the schedule, is that when we try to resolve overtime on a CSC, this can only be done by providing help to that station shortly before or at the moment this overtime occurs (see Section 4.4.4 on candidate CSCs). The testing method applied here is not suited to test the tradeoff between searching in a larger window containing multiple CSCs, and searching in multiple smaller windows more extensively. Therefore, we include the optimization window width in our analysis of the guidance rules to give better insight.

5.6 Optimization of Guidance Rules

We proposed different guidance rules in Section 4.6.1. Because we have a limited CPU time of 30 minutes (1800 seconds) available, we have to assign a portion of this time to each of the 11 working areas. We assign CPU time to each of the areas proportionate to the number of CSCs with overtime that remain after an initial solution is created (see Table 23), using Parallel First Come First Serve with DOS rule 1 and Visit rule 2 (as selected in Section 5.4).

Area	1	2	3	4	5	6	7	8	9	10	11
Floaters	1	2	2	2	2	1	2	1	1	1	1
Avgerage nr. of CSCs left	36	74	404	1274	97	302	911	70	7	68	28
Assigned CPU Time (fraction of total)	0,01	0,02	0,12	0,39	0,03	0,09	0,28	0,02	0,00	0,02	0,01
Assigned CPU Time (in seconds)	20	41	222	701	54	166	501	39	4	38	15

Table 23. Assigned CPU time per Area

The two proposed guidance rules for selecting CSCs are as follows:

1. Regular Guidance: Sort candidate CSCs on overtime with the most overtime on top, and select CSCs from top to bottom. If the last candidate CSC has been reached, and there is still time left, sort again and start from the top.
2. Tabu Guidance: Sort candidate CSCs on overtime with the most overtime on top. Always select the top-most CSC that is not on the Tabu list. If a CSC is selected, it is put on a Tabu list, preventing it from being selected for a number of iterations of *tabumax*.

For both guidance rules, we update the overtime on the candidate CSCs after each partial schedule has been optimized and a candidate CSC is skipped if it has no overtime. The regular guidance rule has no additional parameters. We try to find a Tabu list length for Tabu Guidance that prevents cyclic behavior, selecting the same CSC over and over again. We do this by applying Tabu Guidance for 500 seconds to Area 7 using 1000 ENS and our selected configuration for SA. The optimization window width is set at 5 minutes. We track the number of times each CSC was selected by the guidance rule to investigate cyclic behavior. If a working area contains a smaller number of candidate CSCs with overtime the Tabu list length should be shorter, therefore we define the Tabu list length as (*Initial CSCs with overtime*f*), where *f* is a number between 0 and 1. Setting f to a value of 1, set the Tabu list length to the number of CSCs with overtime, which equals Regular Guidance. Results for different values of *f* are shown in Figure 45.

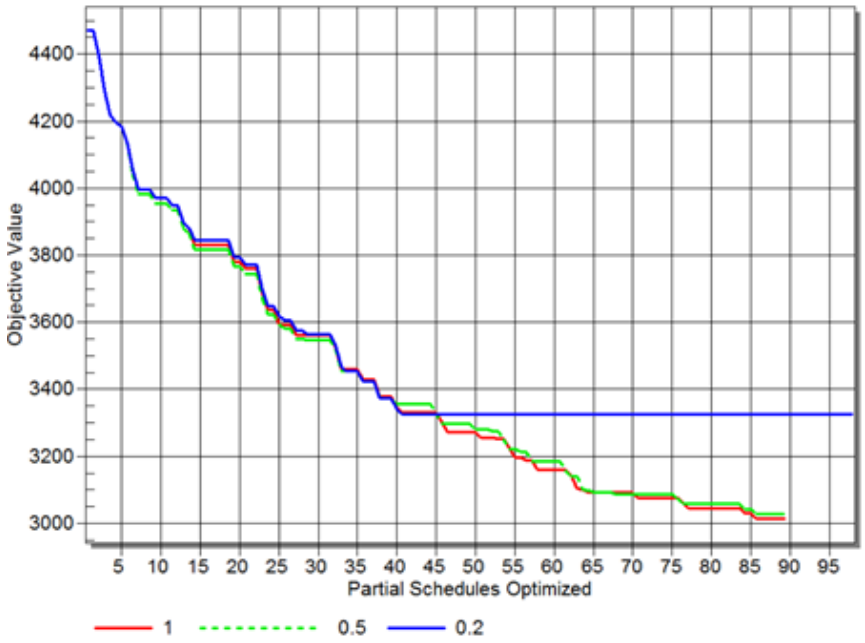


Figure 45. Performance of Tabu Guidance for different Tabu list lengths (values of f)

Cyclic behavior is clearly present for a *f* value of 0.2. A value of 0.5 was enough to prevent this (for both Area 4 and 7). We chose to not further increase this value, to keep Tabu Guidance unique from Regular Guidance. A value of 1 did not change the performance significantly. We noticed some CSCs were being selected multiple times (3 to 4 times) while other CSCs with overtime were not selected at all. This is a result of the Tabu list size being smaller than the original number of CSCs. Also, additional CSCs with overtime might emerge as we optimize partial schedules. The optimization method might choose resolving a large quantity of overtime on one CSC at the cost of two other CSCs having a small amount of overtime.

We want to select the CSC with the most overtime, but in a later stage of the optimization process it might be beneficial to guide the search towards CSCs that have not yet been selected, as they are potential areas of improvement. Therefore we add an additional Tabu element, which prevents a CSC from being selected more than twice.

Final experiment

A final experiment was performed, testing each combination of Guidance Rules, ENS and Optimization window width. Optimization window widths of 3, 5, and 7 minutes were used. Configurations using 500 ENS optimized around 300 partial schedules in the assigned 500 seconds, whereas 1000 and 2000 ENS optimized 220 and 150 partial schedules respectively. Results can be found in Appendix K. The optimization window width was the most influential factor for the performance, with a three minute window width performing best (see Table 24)

Window width	Area 4	Area 7
3 min	2490,86	1670,58
5 min	2574,93	2045,50
7 min	2595,81	2177,28

Table 24. Average ending objective function for different optimization window widths

Regular guidance combined with **1000 ENS** and a **window width of 3 minutes** showed the best performance on both Area 4 and 7, and has therefore been selected as our final setting. 1000 ENS turned out to be the best balance between the number of partial schedules optimized and the quality of each partial schedule, for an optimization window width of 3 minutes. The assigned CPU time was enough to visit candidate CSCs with overtime multiple times (between 2 and 3 times). We note that in applications where the number of problematic CSCs is higher, and the available CPU time is smaller, Tabu Guidance could show better results. Also, for a limited CPU time 500 ENS might show better performance. The reductions found per partial schedule by 500 ENS are smaller, but this can be outweighed by the fact that 1000 ENS takes a longer time per partial schedule. An example is shown in Figure 46, plotting the objective function for 500 and 1000 ENS as function of CPU time.

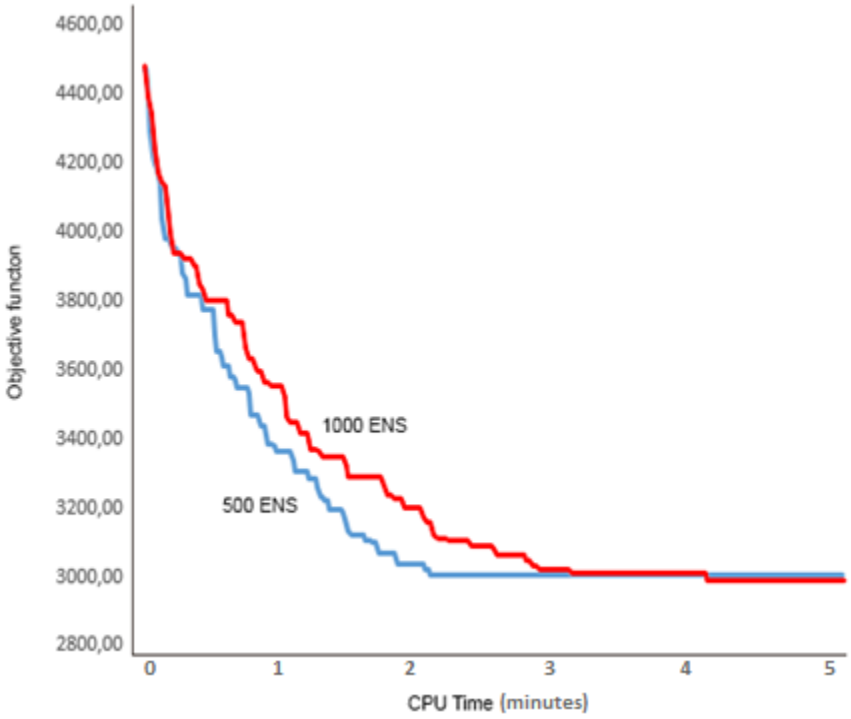


Figure 46. Objective function of 500 vs 1000 ENS, using 5 minutes of CPU time

In the example of Figure 46, it is better so select 500 ENS for the optimization of partial schedules if a CPU time of only 2 minutes is available.

5.7 Total solution configuration

So far, we used a numerical analysis to determine the best performing combination of methods and parameters for the proposed solution method (see Figure 47). We selected Parallel First Come First Serve using DOS rule 1 and Visit rule 2 to create initial solutions. Simulated Annealing using an exponential temperature function was selected to optimize partial schedules. Finally, Regular Guidance was chosen to select CSCs around which the partial schedule will be optimized. The optimization window width has been set to 3 minutes, combined with 1000 Evaluated Neighbor Solutions (ENS) for each partial schedule. During the optimization phase, DOS rule 1 and visit rule 1 are used.

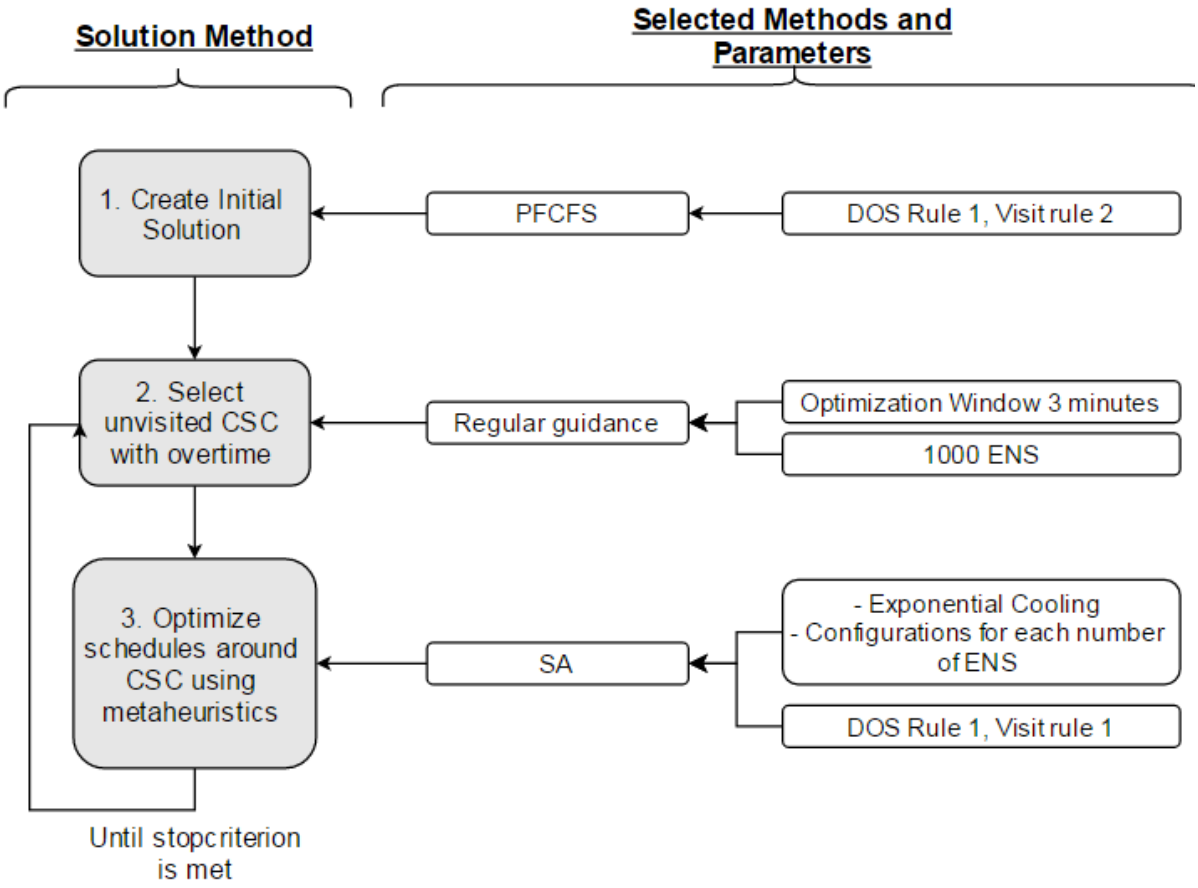


Figure 47. Selected methods and parameters

The process of optimizing parameters and methods has been carried out based on the configuration at Company A and the provided dataset. From our results, we draw the following conclusions regarding the parameters of our solution method:

- In our application, DOS Rule 1 showed the best performance. This means that preemptive floater assistance did not yield additional reductions in overtime and defects. In applications where consecutive problematic CSCs occur more often, we expect preemptive floater assistance (DOS Rule 2 and 3) to become more effective in resolving overtime and defects. Also note that we use the same DOS Rule for all CSCs that are visited by a floater. In cases where overtime on a CSC cannot be resolved by applying DOS Rule 1, it might be beneficial to switch to other DOS Rules. We did not investigate this option.

- In situations where less partial schedules can be optimized (due to limited time and/or lack of processing power), reducing the number of ENS used per partial schedule will yield better results. If excess CPU time is available, increasing the number of ENS will yield better results. The right balance has to be found between the number of partial schedules that is optimized (quantity), and the effort put into each partial schedule (quality).

There is no single best setting for these methods as they are problem dependent. Therefore, the process of setting parameters should be carried every time the solution method is applied at a different company/assembly line. The process described in this chapter can be used as a guideline.

5.8 Benchmark

Now that we have determined the optimal parameters for our solution method, we test its performance on the whole assembly line. The benchmark for the performance of our method is the amount of overtime and defects that remain after applying the heuristic that simulates the current way of working (described in Section 5.3.5). In the current way of working, a floater is assigned to a station when overtime occurs (if a floater is available).

5.9 Total solution performance

We use a total CPU time of 30 minutes (per shift). Using the assignment CPU time as shown in Table 23, and the configuration described in Section 5.7, we create floater schedules for each working area. We then review the total amount of overtime and defects across all working areas. For our solution method, we make a distinction between the overtime and defects that remain after the initial solution is created, and after optimization. Data can be found in Appendix L, and results for one shift are shown in Figure 48.

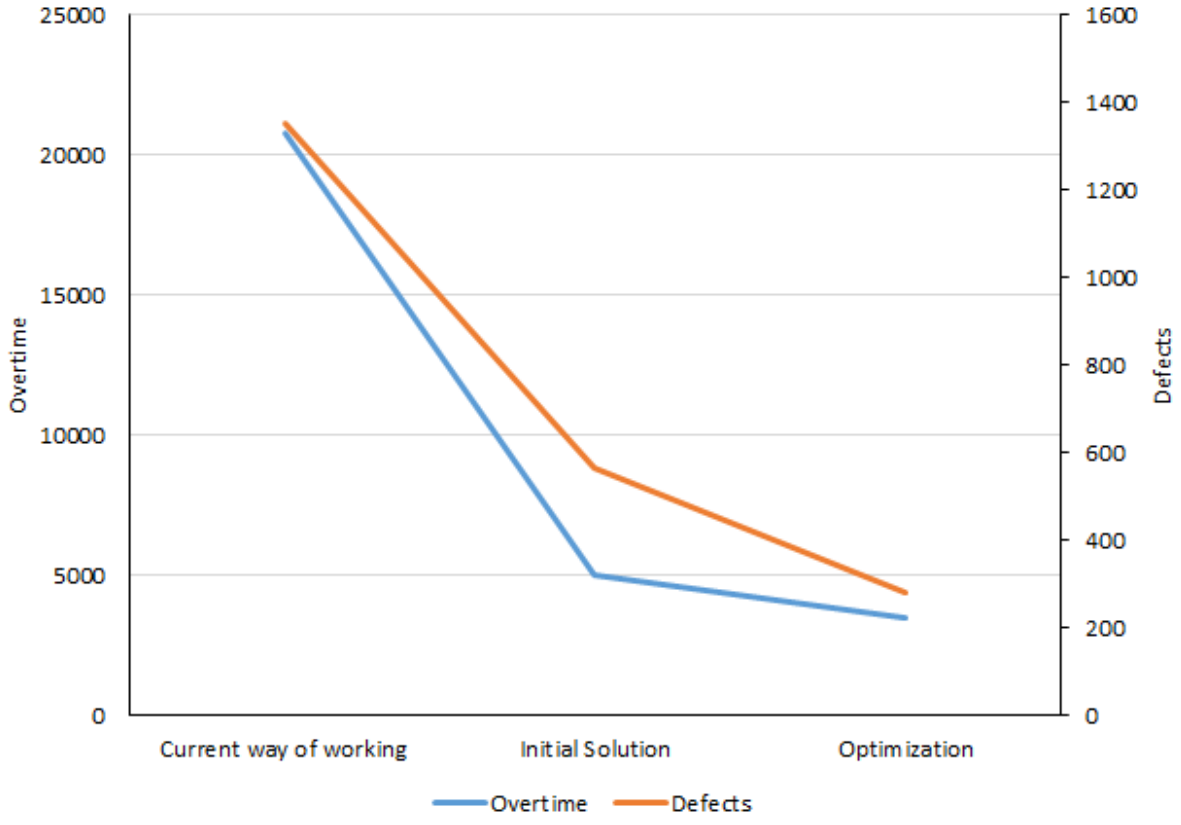


Figure 48. Reduction in overtime and defects

The average amount of overtime and defects at each step of our method, compared to the current way of working, is shown in Table 25.

	Current way of working	Reduction	Initial Solution	Reduction	Optimization	Total Reduction
Overtime	20959,86	→ 77%	4807,53	→ 31%	3333,90	**84%**
Defects	1367,00	→ 57%	586,00	→ 52%	283,50	**79%**

Table 25. Average overtime and defects, reduction per step, and total reduction

The initial solution method realizes a reduction of 77% in overtime and 57% in defects compared to the current way of working. This large step is caused by the fact that in the current way of working, floaters are employed reactively. From the moment a request for help is sent, a floater often does not have enough time to walk to the station to provide help. The initial solution method anticipates overtime, and as a result floaters are employed much more effectively.

The optimization process further improves the floater schedules. A total reduction of 84% in overtime, and 79% in defects is realized compared to the current way of working. This step is smaller compared to the initial solution, as the initial solution picks the 'low hanging fruit' within a matter of seconds. Next, the optimization process has the task of finding improvements that are harder to reach.

5.10 Influence of objective function on overtime and defects

To show the relation between the weights that are chosen for the objective function and the amount of resolved overtime and defects, we run our solution method on Area 7 for 5 minutes, and different values of w2. The results for dataset 1 are shown in Figure 49.

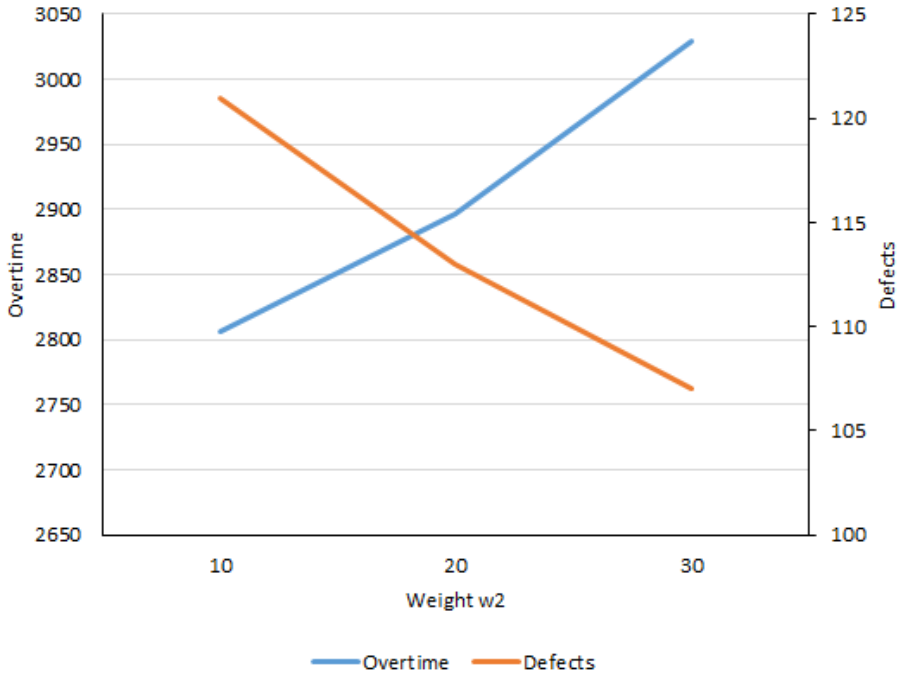


Figure 49. Influence of w2 on overtime and defects

As expected, assigning a higher weight to defects in the objective function will push the solution method to resolve more defects, at the costs of overtime. The weights in the objective function give us the ability to focus the optimization process at the desired quantity (overtime or defects). This is also reflected in the results in Table 25. Because we chose weights of 1 and 10 for overtime and defects respectively, the percentage of resolved defects (51,6%) is higher in the optimization step, compared to overtime (30,7%).

5.11 Conclusion

In this chapter, numerical analysis was used to determine the optimal configuration for out solution method, which is shown in Figure 47. The solution method was tested on the complete assembly line, and showed significant reductions in overtime and defects of 84% and 79% respectively compared to the current way of working.

6. Discussion, Conclusions and Recommendations

In this chapter we first discuss potential areas for improvement for our solution method in Section 6.1. Then we provide conclusions based on our research questions (posed in Chapter 1) in Section 6.2. In Section 6.3, we give recommendations based on the solution method used in this thesis. Finally, we give suggestions for further research in Section 6.4.

6.1 Discussion

In this section, we discuss potential areas for improvement for our solution method.

6.1.1 Problem notation

In Section 4.6.1 we discussed an issue with our problem notation for the metaheuristics that optimize the partial schedules. The notation string only indicates if a CSC should be visited by a floater (and which one) or not. CSCs that occur simultaneously are always visited in the same sequence. To investigate to which extent this occurs, we tracked how many CSCs occur at the same time. On average, a time window of 5 minutes (used in the experiments in Section 5.5.1 and 5.5.2) had 8 different time blocks (takts) in which the candidate CSCs occur. In each of these blocks, there are on average 3 to 4 CSCs present (depending on Area).

To further quantify the influence of CSCs occurring simultaneously, we performed the following experiment. We define a new method, called '*Randomized method*', that works exactly the same as our solution method, but randomizes the sequence of CSCs that occur simultaneously at the start of the optimization process. We let both the standard and randomized methods optimize the same 100 partial schedules. To assure each method is presented with the same information for each partial schedule, we always return to the initial solution after a partial schedule has been optimized. Each method is run 5 times, and we report on the average reduction in objective function per schedule (for each method). Finally, 'Best found' consists of the best partial schedules found by both methods. Results are shown in Table 26.

Area	4	7
Standard	10,17	9,08
Randomized	9,43	9,07
Best found	11,01	9,18
% of Partial schedules for which an improvement was found	9%	2%

Table 26. Impact of sequencing of CSCs that occur simultaneously

The results show that our solution method performs better than a method that uses randomized sequences, but slightly worse than the best found partial schedules. For 9% of the partial schedules, 'Best found' was an improvement compared to the standard method for Area 4. For Area 7, this percentage was 2%.

Despite not changing the sequence in which simultaneously occurring CSCs are visited, our methods are still able to resolve a considerable amount of overtime. We provide two possible explanations:

- Consider three candidate CSCs occurring on different stations at the same moment. In our application, the sequence in which they are visited is determined by their CSC number, with the lowest number being visited first. CSCs numbers are assigned starting with the first station in the line, where CSCs on the first station always have a lower number than the second station, and so on. The working areas mostly have their stations aligned in a straight line (See Figure 5 or Figure 41). In our example, if all three CSCs are assigned to the same floater, the CSC numbers prevent him from first visiting the

station on the far right, then the far left and finally the station in the middle. In this case, the CSC numbers produce a sequence that is favorable to a random sequence.

- The sequencing of CSCs occurring at the same moment has no big impact on the performance of the solution method for the assembly line in this thesis. If distances between stations are such that it is always impossible for a floater to visit more than one station in a single takt, the sequencing of CSCs occurring at the same moment has no impact on performance. In our previous example, visiting one of the three CSCs would then automatically rule out the possibility of visiting the other two CSCs.

We note that the number of CSCs occurring in the same timeframe will increase as the takt time of the line and/or the number of stations in a working area increases. Therefore, sequencing of these CSCs will become more important for assembly lines with longer takts. Starting the optimization process of partial schedules with multiple randomized sequences of CSCs, is one of the multiple options that can be used.

6.1.2 DOS Rules

The Duration of Stay (DOS) Rules proposed in this thesis, only look at the CSC that we are considering to assign to a floater schedule when determining how long a floater should stay at that CSC. These rules could be extended, such that they look at the CSC that follows the CSC under consideration. For example, if a DOS Rule aims to stay longer than is necessary to resolve the overtime on a CSC (DOS Rule 2 and 3), it could check if the CSC that follows it has a delayed start. If this is not the case, staying longer has no use.

6.1.3 Neighbor structure

The optimization methods used in this thesis, do not use complicated neighbor structures. On the one hand, this keeps CPU times low, but on the other hand a more sophisticated approach could be used. For example, if during the translation from problem encoding to floater schedules a CSC is deleted from the schedule to retain feasibility (see Section 4.6.4), this CSC could be marked. One possible neighbor could then be constructed by swapping this and all following CSCs between floaters.

6.1.4 Optimization of parameters

The solution method proposed in this thesis integrates multiple heuristics. In our numerical analysis, we aimed to optimize the multiple parameters associated with these methods. Because of the large number of parameters, not all possible combinations could be tested. This has been solved by using an experiment plan, based on insight in the problem combined with common sense. However, it is unlikely that the perfect combination of parameters has been found, and interactions between parameters such as the optimization window width and the Tabu list length of the guidance rule could pose areas for improvement.

6.2 Conclusions

In Chapter 2 we answered research question 1: '*What does the final assembly at Company A look like?'* We started off by describing the configuration and layout of the assembly line. Next, the characteristics were discussed, such as the takt and station layout. Next we discussed how floaters are employed in the current way of working. Finally, the contents of the planning system that this thesis aims to contribute to were described.

In Chapter 3 we reviewed literature to answer research question 2: '*What methods can be used to schedule floaters in MMALs according to the literature?*' There are no off-the-shelf methods for solving the problem discussed in this thesis. Constructive heuristics for the VRP problem were discussed, as they were deemed useful for creating initial solutions to the problem in this thesis. We concluded that the problem in this thesis is a combinatorial optimization problem, and metaheuristics were described that are used to find solutions to such problems. These metaheuristics are Simulated Annealing, Tabu Search and a Genetic Algorithm.

In Chapter 4 we formulated a solution method, giving an answer to research question 3: *What method is best suitable for minimizing overtime and defects in the assembly line of Company A?* To this end, the functional requirements and a detailed problem description were given in Section 4.1 and Section 4.2. Based on this problem description, a model was formulated in Section 4.3 for which we proposed a solution method in Section 4.5 and Section 4.6. First an initial solution (floater schedule) is made by using a constructive heuristic. Next we apply an optimization process, which iteratively selects a CSC with overtime using *guidance rules*, and optimizes the partial schedule around this CSC using a metaheuristic. We proposed two guidance rules (*Regular* and *Tabu Guidance*), and two metaheuristics (Simulated Annealing and a Genetic Algorithm).

In Chapter 5, we tackled the next research question: 'What are the optimal values for the different parameters of our methods, and how does the total solution method perform? We did this by formulating and executing an experiment plan. First, we identified the best method for creating an initial schedule. Next, we determined the parameters of the metaheuristics for a set number of Evaluated Neighbor Solutions (ENS) to keep results comparable. By running each (configuration of the) metaheuristic on the same partial schedules, the best performing configuration for each number of ENS (500, 1000 and 2000) was identified. SA was selected as best performing metaheuristic for the task of optimizing partial schedules. The number of ENS allowed us to balance the CPU time between the number of partial schedules optimized and the effort put into each schedule. This was used in the search for the best guidance rule. For our application, Regular Guidance combined with 1000 ENS and an optimization window width of 3 minutes showed the best performance.

Our solution method was compared to the current way of working, which was simulated using a constructive heuristic. Significant reductions in overtime and defects were realized. For the total assembly line, an average reduction in overtime and defects of 84% and 79% respectively was achieved using the proposed solution method.

6.3 Recommendations

In this section, we discuss recommendations based on the methods applied in this thesis.

6.3.1 Balancing costs between floaters and overtime

The goal of employing floaters in the assembly line is to reduce overtime. The reasoning being that employing a floater to prevent overtime is better than repairing any non-finished work in a later stage of the assembly line. In practice, assembly lines have built in methods to repair these issues: cars can be taken off the line if they become too problematic; repair stations can be placed in the assembly line to repair defects that have formed; cars can be repaired after they have left the regular assembly line; or the line can be stopped if a station is not able to finish its activities.

All these methods come at a certain cost, and so do floaters. Employing a line manager who is already present as floater might not increase costs, but hiring a dedicated floater does. The method proposed in this thesis aims to schedule the available floaters as efficiently as possible. As someone might expect, each extra floater has diminishing returns. In Figure 50 the amount of overtime and defects in one shift is given for Area 7, as a function of the number of employed floaters (using out solution method).

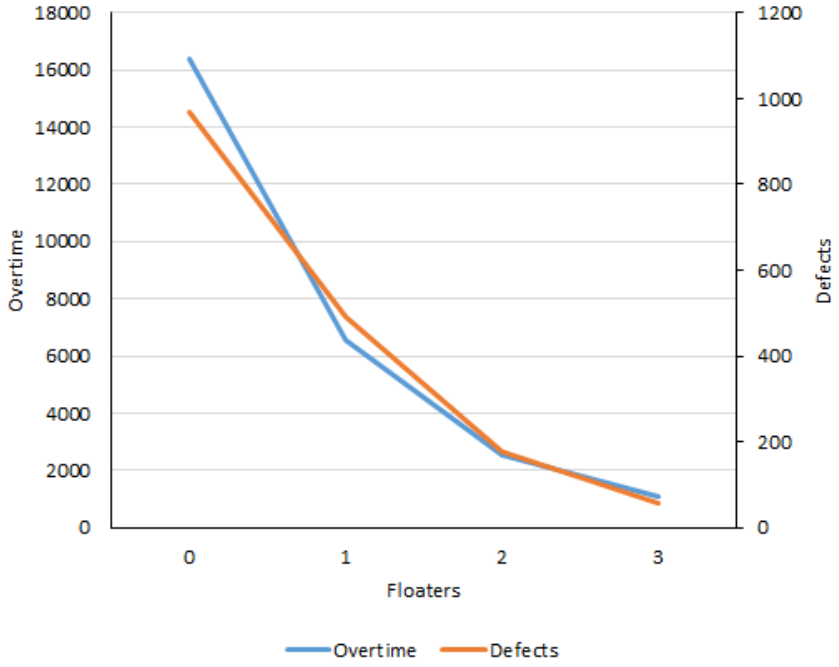


Figure 50. Diminishing returns of each additional floater

In practice, costs have to be balanced between hiring extra floaters and the overtime that they resolve. Due to diminishing returns of employing an extra floater, it might be more beneficial to let this overtime be resolved in a different way. The proposed scheduling methods can help managers make this decision.

6.3.2 Disruptions and stochastic behavior

The planning system, and subsequently the scheduling methods from this thesis, use deterministic processing times. In reality, processing times will fluctuate as workers will not always work at the same pace for different reasons. The scheduling approach in this thesis does not take this into account. If a station can finish their activities on a car just in time in the planning system, there is a chance that in reality this car will cause overtime. As the scheduling methods from this thesis operate offline, and do not receive feedback from the assembly line, these scenarios would have to be planned for. Our proposed solution method could be adjusted to account for these situations in the following way. First we choose a threshold for finishing times. For example, this threshold could be 5 seconds before the exit time of a car at a station. If a station finishes its

activities on a car within these 5 seconds before the car exits the station, this CSC could then be 'flagged' with a new type of overtime, *preemptive overtime*. During the scheduling of floaters, we could try to resolve this preemptive overtime in an additional step after the optimization of floater schedules, or during the optimization by adding a secondary objective that encourages visiting CSCs with preemptive overtime.

As the number of disruptions in the assembly line increases, the performance of the (purely offline) scheduling methods from this thesis will decrease. Also, the relative performance compared to the current way of working (which contains 'online' scheduling rules) will decrease. Floater schedules should in some cases be adapted to real time information. This can be done using simple rules that temporarily override the constructed floater schedules, or using feedback to the planning system and constructing new schedules.

6.3.3 Changes in car sequence during production shift

Floater schedules are produced before a production shift starts. If during the production shift the sequence of cars is changed, this makes the current schedule invalid. Changes in the schedule could be accounted for if quick feedback can be provided to the planning system. Using our methods for creating initial solutions, floater schedules can be created in a matter of seconds for the first cars of the new sequence that enter the assembly line. This could then provide the scheduling methods the required time to create new optimized floater schedules, starting with the first area (see Figure 51). Some working areas might not even need a temporary schedule, as there is enough time to create new schedules before the first car of the changed sequence enters the working area.

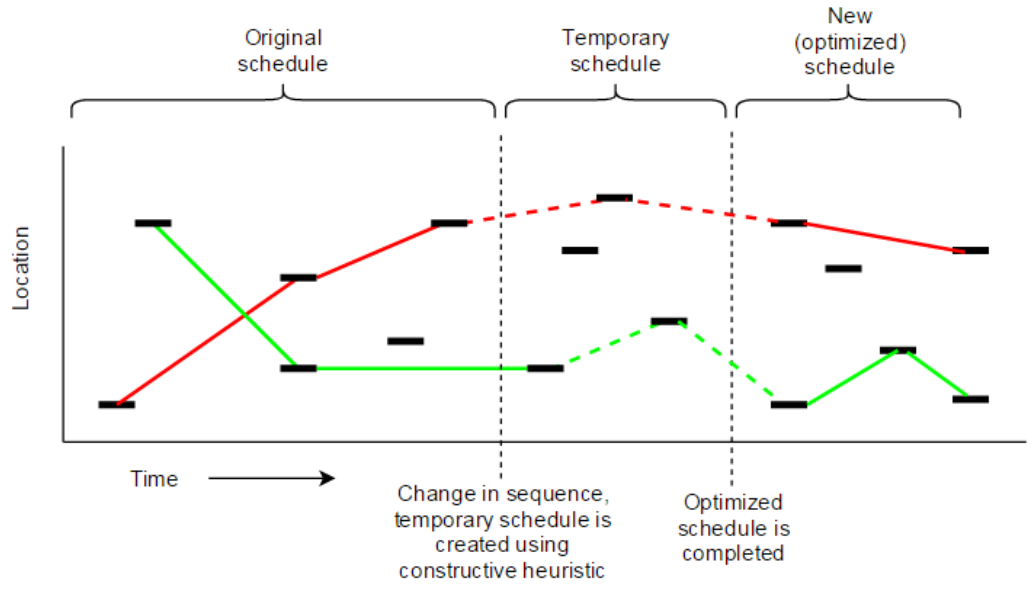


Figure 51. Example of adaptive scheduling

6.3.4 Accuracy limits on scheduling

Just as processing times may vary in reality, so may traveling times of floaters walking through the assembly line, together with their own productivity. The model in this thesis uses a resolution of 1 second, and as a result floaters can be allocated to a CSC for a very short time. One might question the practical use letting a floater work on a certain car for say 4 seconds. The model proposed in this thesis could be adapted to prevent this. In the process of creating floater schedules, a duration of stay is calculated for each CSC visit. We can define a new resolution of for example 10 seconds. The duration of stay can then be calculated as multiples of the resolution by rounding up the fraction (duration of stay/resolution). A stay of 4 seconds would then result in a stay of 10 seconds, and setting the resolution equal to the takt time, would always allocate floaters for the duration of a whole takt. Note that as we increase the resolution, schedules will become less detailed and resolve less overtime in total.

6.4 Suggestions for further research

The solution method proposed in this thesis is a first step towards operational scheduling of floaters in Mixed Model Assembly Lines. In the design of our solution method decisions have been made that pose potential areas for improvement. A more detailed discussion can be found in Section 6.1. Further fine-tuning of parameters and more research into for example the sequencing of CSC occurring simultaneously might further improve performance.

In our application, schedules are produced offline before the production shift starts. We expect modern assembly lines to increasingly become integrated with IT applications. This allows planning systems to react to unforeseen disturbances. Handheld devices with modern communication technology enables planning systems to change floater schedules on the go. Finding efficient methods to schedule floaters in real time (in an uncertain environment), might prove to be a valuable area for further research.

References

Bäck, T. (1993). Optimal Mutation Rates in Genetic Search. *Proceedings of the fifth international conference on genetic algorithms*, 2-8.

Beasley, D., Bull, D. R., & Martin, R. R. (1993). An Overview of Genetic Algorithms: Part 1, Fundamentals. *University Computing, 15(2)*, 58-69.

Blum, C., & Roli, A. (2003). Metaheuristics in Combinatorial Optimization: Overview and Conceptual Comparison. *ACM Computing Surveys 35 (3)*, 268-308.

Bock, S., Rosenberg, O., & Brackel, T. v. (2006). Controlling mixed-model assembly lines in real-time by using distributed systems. *European Journal of Operational Research, 168(3)*, 880-904.

Boysen, N., Fliedner, M., & Scholl, A. (2007). A classification of assembly line balancing problems. *European Journal of Operational Research, 183(2)*, 674-693.

Brucker, P., Drexl, A., Möhring, R., Neumann, K., & Pesch, E. (1999). Resource-constrained project scheduling: Notation, classification, models, and methods. *European Jornal of Operations Research, 112*, 3-41.

Chakraborty, M., & Chakraborty, U. K. (1997). An analysis of linear ranking and binary tournament selection in genetic algorithms. *Proceedings of 1997 International Conference on Information, Communications and Signal Processing, 1*, 407-411.

Deb, K., Pratap, A., Agarwal, S., & Meyarivan, T. (2002). A fast and elitist multiobjective genetic algorithm: NSGA-II. *IEEE Transactions Evolutionary Computation, 6(2)*, 182-197.

Eglese, R. W. (1990). Simulated Annealing: A tool for Operational Research. *European Journal of Operational Research, 46*, 271-281.

Ghosh, S., & Gagnon, R. (1989). A comprehensive literature review and analysis of the design, balancing and scheduling of assembly systems. *The International Journal of Production Research, 27(4)*, 637-670.

Glover, F. (1989). Tabu Search - Part I. *ORSA Journal on Computing, 1(3)*, 190-206.

Goldberg, D. E., & Deb, K. (1991). A Comparative Analysis of Selection Schemes used in Genetic Algorithms. *Foundations of Genetic Algorithms, 1*, 69-93.

Graham, R. L. (1995). *Handbook of combinatorics (Vol. 1).* Elsevier.

Gronalt, M., & Hartl, R. (2003). Workforce planning and allocation for mid-volume truck manufacturing: A case study. *International Journal of Production Research, 41(3)*, 449-463.

Gronalt, M., & Hartl, R. (Technical Report, POM-Working Paper, 2000). *Worker and floater time allocation in a mixed-model assembly line.* Vienna: University of Vienna, Center of Business Studies.

Gujjula, R., & Günther, H.-O. (2009). Scheduling utility workers at mixed-model assembly lines. *Industrial Engineering and Engineering Management*, 1092-1096.

Kirkpatrick, S., Gelatt, C., & Vecchi, M. (1983). Optimization by Simulated Annealing. *Science, 220(4598)*, 671-680.

Krüger, D., & Schöll, A. (2009). A heuristic solution framework for the resource constrained (multi-)project scheduling problem with sequence-dependent transfer times. *European Journal of Operational Research 197* , 492-508.

Mac Duffie, J., Sethuraman, K., & Fisher, M. L. (1996). Product variety and manufacturing performance: evidence from the international automotive assembly plant study. *Management Science, 42(3)*, 350-369.

Mayrhofer, W., März, L., & Sihn, W. (2013). Simulation-based optimization of personnel assignment planning in sequenced commercial verhicle assembly: A software tool for iterative improvement. *Journal of Manufacturing Systems, 32*, 423-428.

Merengo, C., Nava, F., & Pozzetti, A. (1999). Balancing and sequencing manual mixed-model assembly lines. *International Journal of Production, 37(12)*, 2835-2860.

Op Den Kelder, C. (2015). *A planning tool for mixed-model assembly lines in the automotive industry.* Enschede: University of Twente.

Potvin, J., & Rosseau, J. (1995). An exchange heuristic for routeing problems with time windows. *Journal of the Operational Research Society, 46(12)*, 1433-1446.

Scholl, M., Kiel, M., & Scholl, A. (2010). Sequencing mixed-model assembly lines to minimise the number of work overload situations. *International Jounal of Production Research, 49(16)*, 4735-4760.

Solomon, M. (1987). Algorithms for the vehicle routing and scheduling problems with time window constraints. *Operations Research, 35(2)*, 254-265.

Taillard, E., Badeau, P., Gendreau, M., Guertin, F., & Potvin, J. Y. (1997). A tabu search heuristic for the vehicle routing problem with soft time windows. *Transportation science, 31(2)*, 170-186.

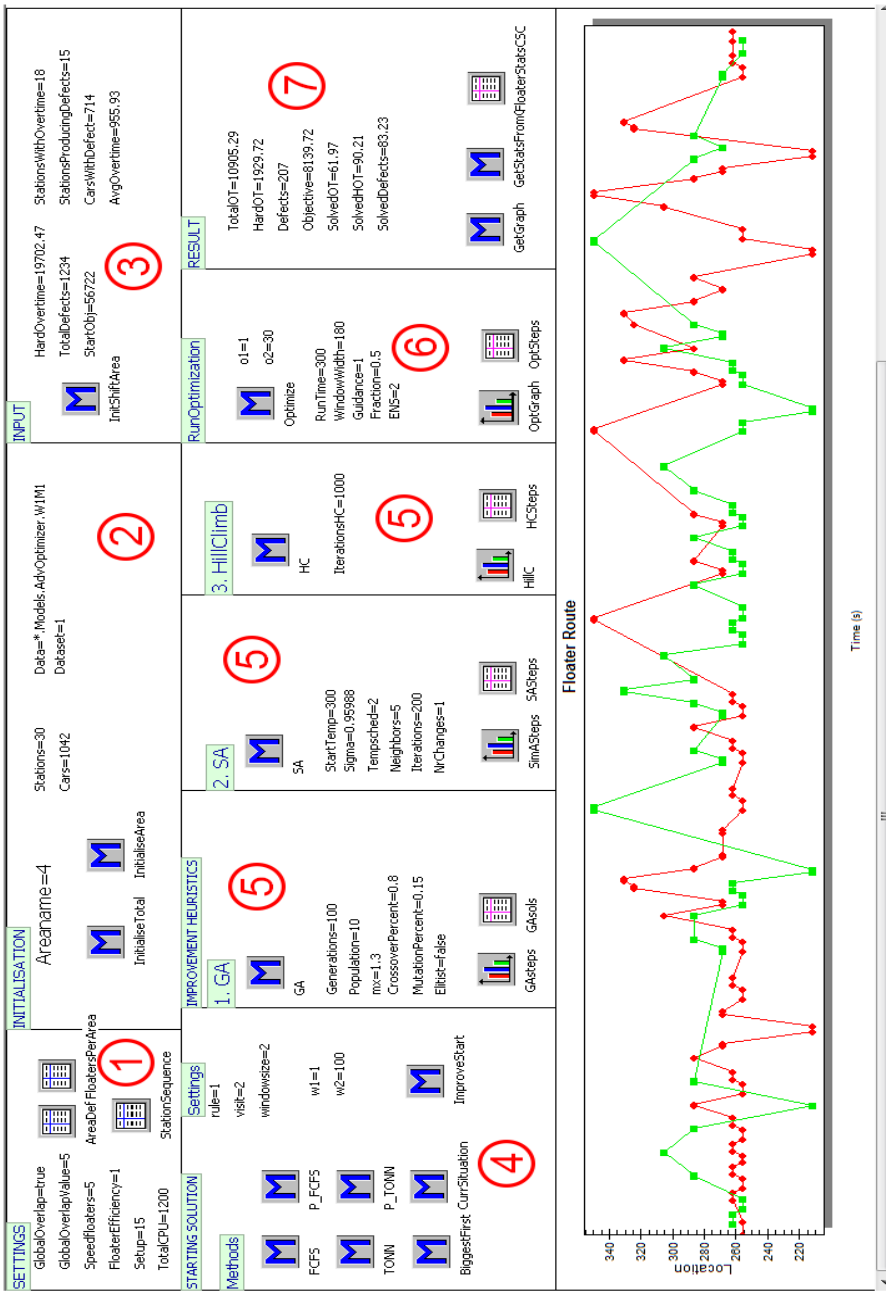
Whitley, D. (1994). A Genetic Algorithm Tutorial. *Statistics and Computing, 4*, 65-85.

Appendices

Appendix A: Abbreviations and vocabulary

MMAL(s) – Mixed Model Assembly Line(s)
FAMMAL – Floater allocation in MMALs
Takt – The cycle time of the paced assembly line, in this thesis it consists of 52.8 seconds.
(Capacity) Overload – More than the available capacity is required at a station for a certain car
Overtime – Amount of time a station would need extra to finish its activities on a certain car
Hard Overtime – Overtime outside the overlap of a station, e.g. unfinished work.
Floaters – Cross trained workers that help in overload situations in MMAL
Imbalanced Line – Line is designed such that inherently some stations have more processing time than others
CSC – Car/Station Combination, every car passes a station only once, and thus a CSC has a fixed time window in which work can be performed on the car.
DOS Rule – Rule that determines the Duration of Stay for a floater at a selected CSC in the initial (constructive) phase of the solution heuristic.
Visit Rule – Rule that determines what happens when a floater cannot visit a CSC for the prescribed duration of stay by the selected DOS rule.
Candidate CSCs – CSCs that have hard overtime on them, or delay the start of a next CSC
Guidance rules – Rules that select a CSC with overtime, around which the floater schedules will be optimized
Optimization window – The window around a selected CSC, in which the floater schedules will be optimized (See Figure 30).
Partial Schedule – The part of the floater schedule that is inside the optimization window
SA – Simulated Annealing
GA – Genetic Algorithm

Appendix B: Screenshot of floater scheduling frame in Plant Simulation



1. Configuration settings, such as overlaps, floater speed, setup time, assigned CPU and floaters assigned per Working Area.
2. Methods for initialization (loading dataset)
3. Starting performance indicators (overtime, defects, starting objective, etc.)
4. Initial solution methods and their parameters
5. Optimization heuristics for partial schedules and their parameters
6. Total optimization method, including parameters for selecting Guidance Rule, optimization window width and number of ENS per partial schedule
7. Results of optimization
8. Bottom, not numbered: Graph showing the floater schedule for the current area for the first 1,5 hour (green = floater 1, red = floater 2).

Appendix C. Results of Hill Climbing algorithm on Working Area 4 and 7

Iterations	A.R.	R. %	CPU	Area
500	14,40	0,41	0,63	4
1000	14,78	0,41	1,24	4
2000	15,11	0,42	2,46	4
500	14,68	0,33	0,70	7
1000	15,18	0,34	1,37	7
2000	15,62	0,34	2,71	7

Iterations: Number of iterations per Partial Schedule.
A.R.: Average Reduction in Objective Function per Partial Schedule.
R.%: Percentage of Partial Schedules for which a reduction was found.
CPU: CPU time per partial schedule in seconds.
Area: Working Area.

Appendix D. Results of Simulated Annealing on Working Area 4

Area 4							
ENS	**N**	**T.F.**	**C**	**A.R.**	**R%**	**CPU**	**#**
500	2	1	1	10,28	47	0,61	13
500	2	1	3	11,08	47	0,62	10
500	2	2	1	21,76	66	0,63	7
500	2	2	3	24,89	78	0,64	1
500	5	1	1	10,96	56	0,62	11
500	5	1	3	10,39	54	0,62	12
500	5	2	1	20,10	70	0,64	8
500	5	2	3	24,49	75	0,64	5
500	10	1	1	9,20	47	0,62	15
500	10	1	3	7,87	35	0,61	16
500	10	2	1	23,45	75	0,64	6
500	10	2	3	24,73	79	0,64	2
500	20	1	1	10,21	50	0,62	14
500	20	1	3	11,36	45	0,62	9
500	20	2	1	24,70	73	0,64	3
500	20	2	3	24,55	78	0,64	4
1000	2	1	1	13,68	57	1,22	14
1000	2	1	3	15,52	63	1,21	9
1000	2	2	1	23,85	74	1,25	8
1000	2	2	3	25,78	75	1,25	4
1000	5	1	1	13,77	55	1,22	13
1000	5	1	3	13,59	67	1,21	15
1000	5	2	1	26,36	77	1,25	1
1000	5	2	3	26,19	75	1,26	2
1000	10	1	1	11,04	47	1,20	16
1000	10	1	3	15,16	61	1,20	10
1000	10	2	1	24,43	77	1,25	6
1000	10	2	3	25,86	77	1,25	3
1000	20	1	1	13,93	53	1,23	12
1000	20	1	3	13,94	55	1,20	11
1000	20	2	1	25,07	71	1,25	5
1000	20	2	3	24,09	74	1,26	7
2000	2	1	1	17,67	71	2,39	12
2000	2	1	3	17,65	62	2,40	13
2000	2	2	1	27,95	78	2,48	3
2000	2	2	3	27,09	79	2,49	5
2000	5	1	1	16,75	62	2,39	16
2000	5	1	3	17,95	70	2,39	11
2000	5	2	1	27,63	75	2,51	4
2000	5	2	3	26,06	80	2,51	7
2000	10	1	1	16,86	64	2,39	15
2000	10	1	3	18,43	63	2,37	10
2000	10	2	1	28,20	77	2,48	1
2000	10	2	3	26,50	77	2,49	6
2000	20	1	1	19,56	70	2,38	9
2000	20	1	3	17,39	71	2,39	14
2000	20	2	1	25,13	71	2,49	8
2000	20	2	3	28,09	82	2,48	2

Abbreviations:

ENS: Evaluated Neighbor Solutions.

N: Neighbors at each temperature.

T.F.: Temperature Function (1 = Linear, 2=Exponential).

C: Initial Number of Changes.

A.I.: Average Reduction in Objective Function per Partial Schedule.

R%: Percentage of Partial Schedules for which a reduction was found.

CPU: CPU time per partial schedule in seconds.

#: Ranking of configuration for its number of ENS

Note: The number of iterations for each configuration is calculated as ENS/N. After each iteration, the temperature is lowered.

Appendix E. Results of Simulated Annealing on Working Area 7

Area 7							
ENS	**N**	**T.F.**	**C**	**A.R.**	**R%**	**CPU**	**#**
500	2	1	1	18,35	45	0,67	13
500	2	1	3	17,86	46	0,67	14
500	2	2	1	21,13	51	0,69	5
500	2	2	3	21,42	48	0,69	3
500	5	1	1	17,49	45	0,67	15
500	5	1	3	18,89	46	0,67	12
500	5	2	1	21,40	48	0,69	4
500	5	2	3	20,82	49	0,69	7
500	10	1	1	15,99	44	0,67	16
500	10	1	3	19,96	43	0,67	9
500	10	2	1	20,89	50	0,69	6
500	10	2	3	22,22	50	0,71	1
500	20	1	1	19,55	46	0,67	10
500	20	1	3	19,14	44	0,67	11
500	20	2	1	20,53	51	0,69	8
500	20	2	3	22,14	52	0,69	2
1000	2	1	1	18,42	51	1,34	16
1000	2	1	3	19,89	50	1,32	15
1000	2	2	1	21,83	53	1,35	6
1000	2	2	3	21,74	51	1,35	7
1000	5	1	1	20,92	48	1,32	13
1000	5	1	3	21,18	49	1,33	10
1000	5	2	1	22,21	53	1,35	2
1000	5	2	3	22,12	52	1,36	3
1000	10	1	1	20,02	51	1,32	14
1000	10	1	3	21,52	49	1,32	8
1000	10	2	1	21,96	51	1,35	4
1000	10	2	3	22,23	49	1,35	1
1000	20	1	1	21,12	47	1,32	11
1000	20	1	3	21,00	48	1,47	12
1000	20	2	1	21,47	50	1,35	9
1000	20	2	3	21,86	52	1,35	5
2000	2	1	1	21,42	48	2,62	14
2000	2	1	3	21,81	49	2,61	12
2000	2	2	1	22,22	49	2,68	9
2000	2	2	3	22,49	50	2,69	3
2000	5	1	1	21,11	49	2,61	16
2000	5	1	3	22,08	51	2,61	11
2000	5	2	1	22,49	50	2,70	2
2000	5	2	3	22,25	51	2,68	7
2000	10	1	1	21,78	50	2,64	13
2000	10	1	3	22,19	49	2,62	10
2000	10	2	1	22,32	50	2,68	6
2000	10	2	3	22,25	51	2,68	8
2000	20	1	1	21,17	52	2,61	15
2000	20	1	3	22,40	52	2,62	4
2000	20	2	1	22,39	53	2,68	5
2000	20	2	3	22,50	52	2,67	1

Abbreviations:

ENS: Evaluated Neighbor Solutions.

N: Neighbors at each temperature.

T.F.: Temperature Function (1 = Linear, 2=Exponential).

C: Initial Number of Changes.

A.R.: Average Reduction in Objective Function per Partial Schedule.

R%: Percentage of Partial Schedules for which a reduction was found.

CPU: CPU time per partial schedule in seconds.

#: Ranking of configuration for its number of ENS

Note: The number of iterations for each configuration is calculated as ENS/N. After each iteration, the temperature is lowered.

Appendix F. Linear Regression result for SA parameter settings on Area 4 and 7

Coefficients[a,b]

Model		Unstandardized Coefficients		Standardized Coefficients	t	Sig.
		B	Std. Error	Beta		
1	(Constant)	-2,369	,962		-2,461	,018
	Neighbors	,008	,032	,008	,241	,811
	Tempsched	11,363	,438	,904	25,955	,000
	InitialChanges	,336	,219	,053	1,533	,133
	ENS	,004	,000	,356	10,231	,000

a. Dependent Variable: AverageIncrease

b. Selecting only cases for which Area = Area 4

c. R^2 = 0,613, Adj. R^2 = 0,596, F-ratio 36,021 (sig. ,000)

Coefficients[a,b]

Model		Unstandardized Coefficients		Standardized Coefficients	t	Sig.
		B	Std. Error	Beta		
1	(Constant)	16,158	,556		29,081	,000
	Neighbors	,025	,018	,117	1,372	,177
	Tempsched	1,651	,253	,555	6,532	,000
	InitialChanges	,287	,126	,193	2,272	,028
	ENS	,001	,000	,575	6,774	,000

a. Dependent Variable: AverageIncrease

b. Selecting only cases for which Area = Area 7

c. R^2 = 0,690, Adj. R^2 = 0,661, F-ratio 23,899 (sig. ,000)

Appendix G. Optimization of Parameters for Genetic Algorithm

We use the same approach used for SA to optimize the parameters of our Genetic Algorithm. The following parameters can be changed for our GA method:

- Number of generations (iterations).
- Population size.
- Value of *max* between 1 and 2 (adjustment of selection pressure).
- $P_{crossover}$ (chance of performing a crossover operation on any set of parents).
- $P_{mutation}$ (mutation rate, chance of any part of any offspring to become mutated).
- Classical or Elitist approach
- *InsertBest* (Inserting the best solution found so far into the next generation or not).

Determining the value of InsertBest

First we determine if it is profitable to always insert the best solution found so far into the next generation. Therefore we run a preliminary experiment with the following settings. We choose a mutation rate of 1/l where l is the notation string length, a crossover percentage of 80% and a value of 1.9 for max (selection pressure). Finally, we set the number of generations to 33 and a population size of 30 (giving approximately 1000 ENS). In Figure 52, the best objective function is shown per generation for the Classical and Elitist approach for Area 4, where the use of *InsertBest* is indicated with a plus sign.

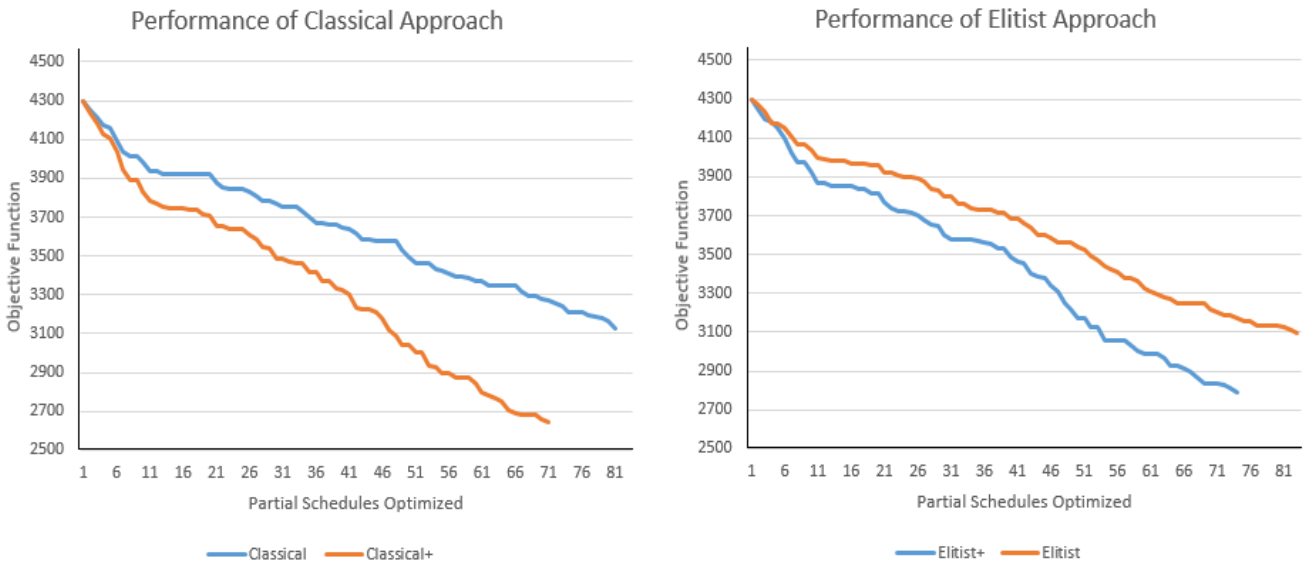


Figure 52. Performance of Classical and Elitist approach with (+) and without the InsertBest element

For each approach, the performance is increased by inserting the best solution found so far into the next generation. These results have also been observed for Area 7. Therefore we apply InsertBest for the remainder of our numerical analysis of GA.

Note: Configurations that show inferior performance run on for longer in the graphs due to the following reason. Our testing framework runs each configuration on the first 100 candidate CSCs with overtime. The ranking of those CSCs is kept fixed, to assure all configurations are run on the same CSCs/partial schedules. Configurations that produce good partial schedules also reduce the overtime on other CSCs in the list of 100 (which is fixed). A CSC in the list that originally had overtime, may not have overtime by the time it is selected, and will therefore be skipped.

Experiment Setup

The mutation rate is calculated automatically using 1/l, where l is the notation string length (number of CSCs

present in the optimization window). We use the same amount of ENS as used for Simulated Annealing, to keep the results comparable. For population sizes, we selected 4 levels: 10, 20, 30, and 40 individuals. These numbers are higher overall than the number of neighbors N for Simulated Annealing, as GAs recombine members of the population to create new solutions and the population should be significantly large to allow diversity. From the three levels of ENS (500, 1000 and 2000) the number of generations is calculated *as ENS/population size*, which is rounded to an integer. For each configuration we test the two selection approaches (Classical and Elitist) and two levels of selection pressure: high (max=1.9) and low (max=1.3). The low value is chosen higher than 1 as a value of 1 would result in equal selection probabilities for all individuals, regardless of rank. For now we hold $P_{crossover}$ fixed at 0.8. The described experiment setup results in a total of 4 experimental factors and a total of 48 different configurations, as shown in Table 27.

ENS	Population Size N	Apprach	Selection Pressure
500/1000/2000	10/20/30/40	Classical/Elitist	1.9/1.3

Table 27. Experimental factors and settings for GA

Again, using the framework discussed in Section 5.3.3, each configuration is applied to optimize the same 100 partial schedules. This is done for Area 4 and 7, and we report on the same performance characteristics as for Simulated Annealing.

Results

The average reduction in objective function per number of ENS for the all configurations of GA is given in Table 28.

		Area 4	Area 7
ENS	500 ENS	13,98	12,10
	1000 ENS	16,95	12,51
	2000 ENS	19,29	12,99

Table 28. Average Reduction in objective function using GA for different numbers of ENS

Again, a larger number of ENS results in better partial schedules on average. The best performing GA configurations for Area 4 and 7 are given in Table 29 and 30 (Results for all configurations can be found in Appendix H and I).

ENS	Population	Generations	Elitist	Pressure	A.I.
2000	20	100	False	1,90	23,23
1000	10	100	True	1,30	20,19
500	10	50	False	1,90	17,10

Table 29. Best performing GA configurations on Area 4

ENS	Population	Generations	Elitist	Pressure	A.I.
2000	30	67	False	1,90	13,95
1000	10	100	False	1,30	13,67
500	10	50	False	1,30	13,54

Table 30. Best performing GA configurations on Area 7

Before we further analyze how the different parameters influence the performance of the GA, we note that the observed performance of GA is considerably worse than SA. On Area 7, the GA is outperformed by our reference Hill Climbing algorithm by approximately 10%. We therefore try to improve the performance of the GA on Area 7 first.

We suspect that the GA algorithm converges too quickly on Area 7. We observed the population during each generation while running the GA, and noticed that the population became filled with individuals that were very similar (homogenous population) in a very early stage (at approximately 20% of generations). The convergence behavior in our experiment is determined by the selection pressure and whether or not an elitist approach is used. In Table 31, the average reduction per partial schedule for each combination of approach and selection pressure is shown for Area 7.

		Pressure	
		Low	High
Elitist	No	13,12	13,13
	Yes	13,01	12,57

Table 31. Average reduction per approach and selection pressure

These results suggest that when an elitist approach is used, it should be coupled with a low selection pressure to keep diversity in the population and prevent the GA from converging too quickly. Also, the non-elitist approach outperforms the elitist approach for both levels of selection pressure. The elitist approach inherently has a higher tendency to converge. When creating a new generation, the current population is added to the newly created offspring. The best 50% is then selected to be put into the next generation. If the generated offspring fails to have a good fitness function (relative to their parents), they will not reach the next generation. Another factor that determines convergence behavior that we have not yet investigated is the mutation rate. We suspect the mutation rate is too small to sufficiently cover the solution space, especially for Area 7. We selected a mutation rate of 1/l, which on average will mutate one 'gene' of each individual in the offspring. This means that no matter how many genes an individual has, the number of genes mutated on average is always 1. Therefore, the percentage of genes (CSCs) mutated per individual will decrease as their notation string length increases. For Area 4, each gene has a probability of approximately (1/21≈) 5% to be mutated, whereas for Area 7 this is (1/33≈) 3%. For applications in which more running time is available, this would be less problematic. In our application, we expect that a fixed mutation rate, higher than 3%, yields better performance on Area 7.

Improving GA performance

By applying a One Factor at A Time approach we try to find a configuration with 1000 ENS for which the performance of GA is at least better than the Hill Climbing algorithm, and closer to (or better than) the performance of Simulated Annealing. We selected a medium population size of 20 to perform the next experiment. Based on the results from Table 31, we use a non-elitist approach coupled with a high selection pressure. We vary the mutation rate (between 3% and 10%) and the selection pressure (between 1.1 and 1.9). After considering 12 different configurations, the average reduction in objective function per partial schedule still did not exceed 14 (see Appendix J). Mutation rate, crossover probability and population size were changed but did not increase performance to acceptable levels. Neither did removing the InsertBest element, or feeding the algorithm a more randomized initial population.

Conclusion Genetic Algorithm

We conclude that the implemented Genetic Algorithm is not a suitable choice for optimizing the partial schedules in our application. This might result from the fact that the Genetic Algorithm is less likely to perform multiple mutations on the same solution. Simulated Annealing (and the Hill Climbing algorithm) make 'a lot of changes to a single solution' in every iteration, whereas the GA makes 'a few changes to a lot of solutions'. The process of GAs should be designed such that these changes can be preserved (diversity in the population), so new and better solutions can be found by recombining and mutating individuals multiple times. But despite our efforts to introduce and maintain diversity in the populations, the populations converged around sub optimal solutions, and GA was unable to efficiently search the solution space in our application.

Appendix H. Results of Genetic Algorithm on Working Area 4

Area 4							
ENS	**Pop**	**E**	**Pres**	**A.R.**	**R%**	**CPU**	**#**
500	10	2	1,90	16,37	56	0,69	3
500	10	1	1,90	17,10	61	0,69	1
500	10	2	1,30	15,92	57	0,69	4
500	10	1	1,30	16,71	65	0,67	2
500	20	2	1,90	14,30	53	0,70	9
500	20	1	1,90	13,79	58	0,66	13
500	20	2	1,30	14,29	55	0,70	10
500	20	1	1,30	13,80	59	0,67	12
510	30	2	1,90	14,43	59	0,72	7
510	30	1	1,90	13,37	65	0,67	14
510	30	2	1,30	15,71	64	0,72	5
510	30	1	1,30	14,33	61	0,67	8
520	40	2	1,90	14,62	63	0,74	6
520	40	1	1,90	13,15	61	0,68	15
520	40	2	1,30	14,27	64	0,75	11
520	40	1	1,30	12,70	67	0,68	16
1000	10	2	1,90	17,95	56	1,35	5
1000	10	1	1,90	20,11	66	1,33	2
1000	10	2	1,30	20,19	66	1,36	1
1000	10	1	1,30	19,62	67	1,31	4
1000	20	2	1,90	17,51	65	1,35	7
1000	20	1	1,90	20,03	63	1,29	3
1000	20	2	1,30	17,84	68	1,35	6
1000	20	1	1,30	16,64	65	1,29	10
990	30	2	1,90	16,93	59	1,35	8
990	30	1	1,90	16,32	65	1,29	12
990	30	2	1,30	15,96	64	1,41	14
990	30	1	1,30	15,41	63	1,29	16
1000	40	2	1,90	16,83	63	1,37	9
1000	40	1	1,90	16,16	70	1,29	13
1000	40	2	1,30	15,90	59	1,38	15
1000	40	1	1,30	16,56	68	1,30	11
2000	10	2	1,90	19,54	68	2,70	15
2000	10	1	1,90	21,63	72	2,60	4
2000	10	2	1,30	20,25	66	2,71	10
2000	10	1	1,30	22,45	71	2,59	3
2000	20	2	1,90	20,18	67	2,66	11
2000	20	1	1,90	23,23	74	2,57	1
2000	20	2	1,30	21,10	72	2,71	7
2000	20	1	1,30	22,82	82	2,58	2
2010	30	2	1,90	17,52	63	2,68	16
2010	30	1	1,90	20,83	74	2,58	8
2010	30	2	1,30	19,70	70	2,68	14
2010	30	1	1,30	21,51	74	2,57	5
2000	40	2	1,90	20,05	65	2,65	12
2000	40	1	1,90	20,52	75	2,58	9
2000	40	2	1,30	21,21	75	2,71	6
2000	40	1	1,30	19,93	75	2,56	13

Abbreviations:

ENS: Evaluated Neighbor Solutions.

Pop: Population Size.

E: Elitist approach (1=no,2=yes)

Pres: Selection pressure.

A.R.: Average Reduction in Objective Function per Partial Schedule.

I.%: Percentage of Partial Schedules for which a reduction was found.

CPU: CPU time per partial schedule in seconds.

#: Ranking of configuration for its number of ENS

Note: Number of generations is calculated as ENS/Population size

Appendix I. Results of Genetic Algorithm on Working Area 7

Area 4

ENS	Pop	E	Pres	A.R.	R%	CPU	#	ENS	Pop	E	Pres	A.R.	R%	CPU	#
500	10	2	1,90	11,68	55	0,75	16	2000	30	2	1,90	12,88	53	2,91	14
500	10	1	1,90	12,24	52	0,74	14	2000	30	1	1,90	13,95	55	2,83	1
500	10	2	1,30	13,04	51	0,76	3	2000	30	2	1,30	13,12	53	2,97	13
500	10	1	1,30	13,54	56	0,72	1	2000	30	1	1,30	13,82	55	2,80	4
500	20	2	1,90	13,10	53	0,75	2	2000	40	2	1,90	13,20	51	2,90	11
500	20	1	1,90	12,62	54	0,72	5	2000	40	1	1,90	13,38	52	2,78	9
500	20	2	1,30	12,64	52	0,76	4	2000	40	2	1,30	13,86	56	2,90	3
500	20	1	1,30	12,48	54	0,73	10	2000	40	1	1,30	13,41	53	2,78	8
510	30	2	1,90	12,62	51	0,78	6								
510	30	1	1,90	11,85	55	0,74	15								
510	30	2	1,30	12,55	54	0,78	7								
510	30	1	1,30	12,42	56	0,73	11								
520	40	2	1,90	12,36	51	0,81	12								
520	40	1	1,90	12,51	56	0,75	9								
520	40	2	1,30	12,51	55	0,81	8								
520	40	1	1,30	12,36	53	0,74	13								
1000	10	2	1,90	11,94	54	1,46	15								
1000	10	1	1,90	13,44	54	1,42	4								
1000	10	2	1,30	13,23	53	1,48	6								
1000	10	1	1,30	13,67	54	1,41	1								
1000	20	2	1,90	12,41	51	1,45	14								
1000	20	1	1,90	13,29	53	1,42	5								
1000	20	2	1,30	12,95	52	1,50	10								
1000	20	1	1,30	12,44	55	1,40	13								
990	30	2	1,90	11,64	53	1,46	16								
990	30	1	1,90	13,49	53	1,39	2								
990	30	2	1,30	13,19	52	1,48	8								
990	30	1	1,30	12,91	55	1,39	11								
1000	40	2	1,90	13,18	54	1,49	9								
1000	40	1	1,90	13,48	55	1,41	3								
1000	40	2	1,30	12,80	55	1,49	12								
1000	40	1	1,30	13,22	55	1,40	7								
2000	10	2	1,90	12,66	53	2,87	15								
2000	10	1	1,90	13,77	53	2,84	5								
2000	10	2	1,30	12,48	53	2,92	16								
2000	10	1	1,30	13,23	54	2,80	10								
2000	20	2	1,90	13,18	51	2,89	12								
2000	20	1	1,90	13,57	52	2,79	7								
2000	20	2	1,30	13,72	55	2,89	6								
2000	20	1	1,30	13,92	56	2,80	2								

Abbreviations:

ENS: Evaluated Neighbor Solutions.

Pop: Population Size.

E: Elitist approach (1=no,2=yes)

Pres: Selection pressure.

A.I.: Average Reduction in Objective Function per Partial Schedule.

I.%: Percentage of Partial Schedules for which a reduction was found.

CPU: CPU time per partial schedule in seconds.

#: Ranking of configuration for its number of ENS

Note: Number of generations is calculated as ENS/Population size

Appendix J. Iterations of improving GA

Generations	Population	Mx	Elitist	A.R.	Changes
50,00	20,00	1,90	FALSE	12,79	6% mutation
50,00	20,00	1,90	FALSE	12,76	10% mutation
50,00	20,00	1,30	FALSE	13,38	1.3 Pressure
50,00	20,00	1,10	FALSE	13,43	1.1 Pressure
50,00	20,00	1,10	FALSE	11,49	No InsertBest
50,00	20,00	1,10	FALSE	13,47	Insertbest, crossover 0.9
100,00	10,00	1,10	FALSE	13,32	10 neighbors
34,00	30,00	1,10	FALSE	13,48	30 neighbors
34,00	30,00	1,10	FALSE	13,01	6% percent mutation
34,00	30,00	1,10	TRUE	12,16	10% mutation, Elitist approach
34,00	30,00	1,10	FALSE	13,39	Classical approach, 15% mutation
34,00	30,00	1,10	FALSE	13,23	50 instead of 100% of genes mutated in initial population

Appendix K. Experiment for Guidance Rules

Start	End	WW	Guid	ENS	Area	Start	End	WW	Guidance*	ENS	Area
3665,62	2475,31	3	1	500	4	4646,52	1655,61	3	1	500	7
3665,62	2455,58	3	1	1000	4	4646,52	1622,51	3	1	1000	7
3665,62	2534,00	3	1	2000	4	4646,52	1688,52	3	1	2000	7
3665,62	2495,44	3	2	500	4	4646,52	1763,95	3	2	500	7
3665,62	2457,49	3	2	1000	4	4646,52	1657,50	3	2	1000	7
3665,62	2527,34	3	2	2000	4	4646,52	1635,38	3	2	2000	7
3665,62	2575,45	5	1	500	4	4646,52	2028,92	5	1	500	7
3665,62	2533,49	5	1	1000	4	4646,52	2040,73	5	1	1000	7
3665,62	2638,06	5	1	2000	4	4646,52	2087,23	5	1	2000	7
3665,62	2587,49	5	2	500	4	4646,52	2050,63	5	2	500	7
3665,62	2524,30	5	2	1000	4	4646,52	2023,91	5	2	1000	7
3665,62	2590,81	5	2	2000	4	4646,52	2041,57	5	2	2000	7
3665,62	2552,56	7	1	500	4	4646,52	2121,62	7	1	500	7
3665,62	2566,55	7	1	1000	4	4646,52	2119,74	7	1	1000	7
3665,62	2651,11	7	1	2000	4	4646,52	2226,67	7	1	2000	7
3665,62	2564,15	7	2	500	4	4646,52	2162,12	7	2	500	7
3665,62	2612,41	7	2	1000	4	4646,52	2237,20	7	2	1000	7
3665,62	2628,06	7	2	2000	4	4646,52	2196,34	7	2	2000	7

Start: Start Objective value
End: End Objective value
WW: Optimization Window Width in minutes
Guid: 1=Regular Guidance, 2=Tabu Guidance

Appendix L. Results for complete solution method on total assembly line

Dataset/Shift 1

Area	**1,00**	**2,00**	**3,00**	**4,00**	**5,00**	**6,00**	**7,00**	**8,00**	**9,00**	**10,00**	**11,00**
OT	64,54	2511,57	4856,69	19702,47	2871,76	5404,83	16379,41	0,00	0,00	463,23	99,88
D	20,00	69,00	413,00	1234,00	105,00	361,00	970,00	0,00	0,00	74,00	34,00
OTCurr	0,00	327,31	170,61	4804,23	451,89	157,59	14867,20	0,00	0,00	4,80	0,00
Dcurr	0,00	51,00	16,00	517,00	15,00	30,00	722,00	0,00	0,00	1,00	0,00
OTInitial	0,00	53,44	116,48	1362,91	230,88	147,36	3085,68	0,00	0,00	4,80	0,00
Dinitial	0,00	4,00	15,00	232,00	12,00	28,00	272,00	0,00	0,00	1,00	0,00
OTOpti	0,00	53,44	76,34	550,65	230,88	14,66	2552,10	0,00	0,00	0,00	0,00
Dopti	0,00	4,00	1,00	85,00	12,00	2,00	178,00	0,00	0,00	0,00	0,00

Dataset/Shift 2

Area	**1,00**	**2,00**	**3,00**	**4,00**	**5,00**	**6,00**	**7,00**	**8,00**	**9,00**	**10,00**	**11,00**
OT	85,34	2576,12	5437,48	20802,91	2893,90	5673,47	16251,49	0,00	39,48	460,19	171,36
D	28,00	70,00	478,00	1412,00	105,00	369,00	996,00	0,00	15,00	67,00	46,00
OTCurr	0,00	326,81	180,76	5317,65	447,54	291,24	14572,09	0,00	0,00	0,00	0,00
Dcurr	0,00	52,00	13,00	539,00	13,00	41,00	724,00	0,00	0,00	0,00	0,00
OTInitial	0,00	63,44	102,95	1349,08	240,88	386,50	2470,65	0,00	0,00	0,00	0,00
Dinitial	0,00	4,00	7,00	291,00	12,00	49,00	245,00	0,00	0,00	0,00	0,00
OTOpti	0,00	63,44	76,34	588,59	240,88	118,04	2102,43	0,00	0,00	0,00	0,00
Dopti	0,00	4,00	1,00	95,00	12,00	9,00	164,00	0,00	0,00	0,00	0,00