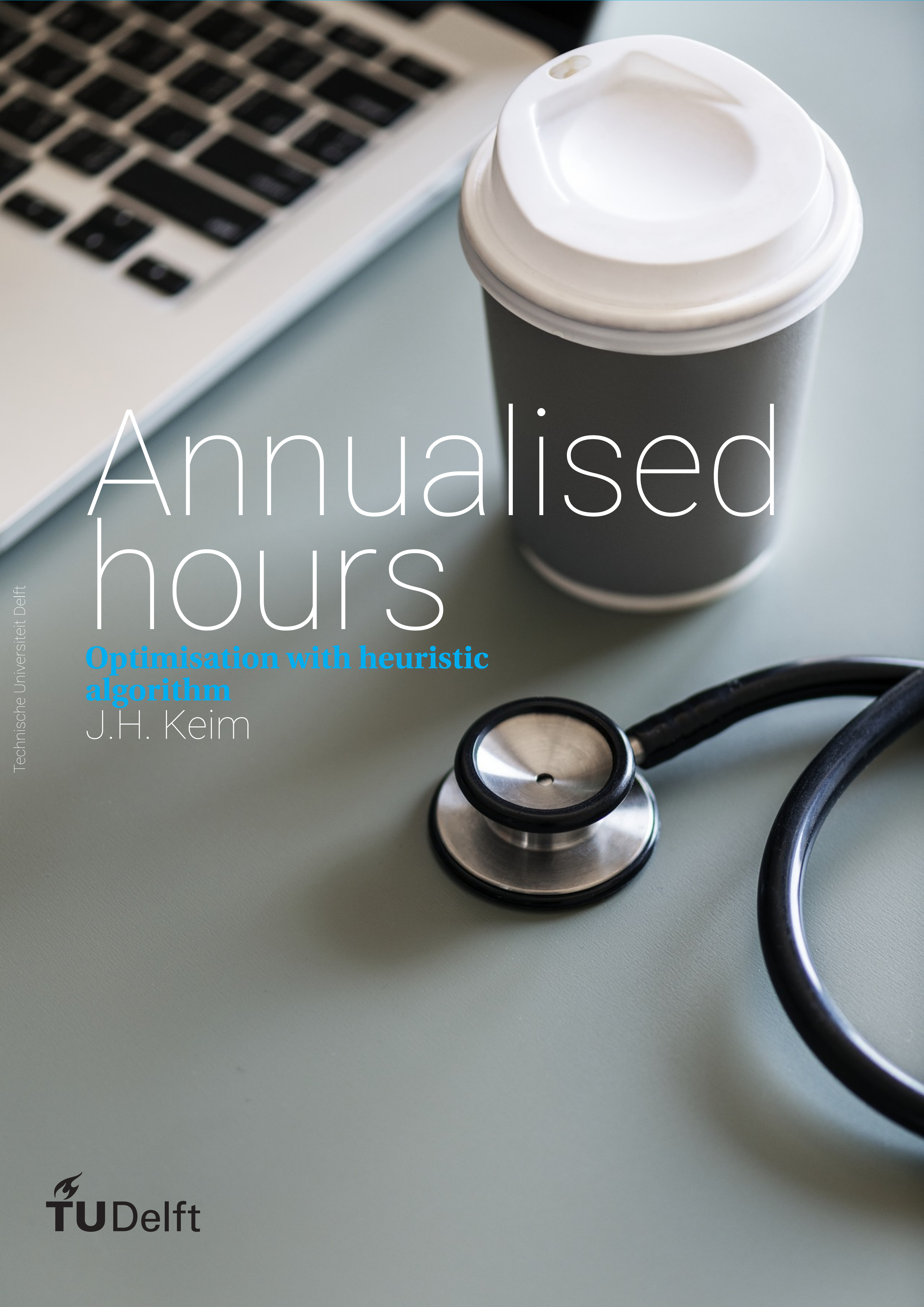

Annualised hours

Optimisation with heuristic algorithm

J.H. Keim

Annualised hours

Optimisation with heuristic algorithm

by

J.H. Keim

to obtain the degree of Bachelor of Science
at the Delft University of Technology,
to be defended publicly on Thursday 5 juli, 2018 at 14:00 PM.

Student number:	4481127	
Project duration:	April 23, 2018 – Juli 5, 2018	
Thesis committee:	Prof. dr. ir. J. T. van Essen,	TU Delft, supervisor
	Dr. K.P. Hart,	TU Delft
	Dr. J. G. Spandaw,	TU Delft

Abstract

In this thesis, we analyse the optimisation of an annualised hours problem. Annualised hours problems are a widely studied subjects, in which the working hours for a given number of employees per week are minimised. We approach this problem with a mixed integer linear program, where the objective function of this problem is divided into two parts; the first is to find the minimisation on the maximum over and under staffing per week, the second to find working hours for each employee as close as possible to their contract hours. Additionally, we have a number of restrictions on the duration of a shift and the minimum and maximum amount of shifts in a given period.
The project is done in collaboration with the company ORTEC. ORTEC provided a random data generator, to test the model with different methods. The model is intended for application on a dataset of 100 employees and 52 weeks. However, we test the model on different data sets of 52 weeks with 50, 100, 150, 200, 250, 1000 employees and for 100 employees with 26, 52, 78 weeks.
Besides analysing the model, we compare different algorithms. We use a Gurobi solver for an exact solution. Further, we consider Lagrangian relaxation, a heuristic method. In this method one of the constraints is put in the objective function. Besides this we consider a heuristic of hill climbing algorithm in combination with different local search algorithms. For the local search we use two neighbourhoods, one based on the employees, the other based on the weeks.
We found that a Gurobi solver solves the model in reasonable computation time with an average of 6 seconds for a data set of 100 employees and 52 weeks. For Lagrangian relaxation, we found an even better computation time than with the Gurobi solver. However, this difference is hardly notable for a data set of 100 employees and 52 weeks and the effect of Lagrangian relaxation is more clear for bigger data sets. The hill climbing algorithm had a consistent and low computation time, but the objective function value differs considerably from the exact solution and resulted in most cases a difference of more than 10 percent.

Preface

This is the bachelor thesis in behave of obtaining the Bachelor in Applied Mathematics at the Delft University of Technology. This project is done at the department of optimisation under supervision of J. T. van Essen and for the company ORTEC under guidance of E. van Veen.

The project exists out of formulating a mixed integer linear program, constructing an algorithm for solving the problem. Where this is an algorithm with control over over the computation time. Further, the project contains the implementation of different algorithms and comparing these on different data sets. For more information about the specific used data and the individual result of each data set there is the possibility to contact me.

I specifically would like to thank my supervisor J. T. van Essen for her tremendous help and support during the project. In addition, I would like to thank E. van Veen. for providing the project and the data and K.P. Hart and J. G. Spandaw for participating in my thesis committee.

J.H. Keim
Delft, July 2018

Contents

1 Introduction

When observing a company, there is an extensive organisation necessary for managing the workforce. There are employees with all kind of contracts: full time, part time and subcontractors. Next to this an organisation has to take into account that each period has a different kind of workload. There are periods with a higher workload and others lower.
Moreover, employees are entitled to take days off. Therefore, there are weeks in which employees are absent and cannot be scheduled.

Since the twentieth century there are all kind of rules around working hours. In the Netherlands, it started in 1911, when a law was accepted of working a maximum of ten working hours a day. Not long after this, in 1919, this was reduced to 8 hours a day and 45 hours a week. In 1961 this was again reduced to 40 hours a week. [1]. In the last decades of the twentieth century, the first rules around annualised hours were introduced. An annualised hours contract contains an average amount of hours that an employee has to work in a given time period. This results in that an employee with a 40 hours contract can work 38 hours in one week and 42 hours in another. Annualised hours contract are a good way to cope with fluctuating demand.

This brings us to the annualised-hours problem; In a contract with annualised hours, the contract hours are not measured in weeks, but instead in a different time period such as a number of months or a year. This gives an organisation the flexibility to have an employee work more in a period where there is a high workload and work less in a period with a low demand. To not have too much working hours in a week, contracts contain a minimum and maximum amount of working hours per week and have rules on minimum and maximum shifts in a given time period.

In this thesis, we present a model which formulates the optimisation of working hours for each employee each week. Optimal in this case means that the working hours per week are as close as possible to the contract hours, while also minimising the over and under staffing.

The project is done for the company ORTEC, a major supplier of software products and consulting in the field of advanced planning and scheduling. They gave the task to develop an algorithm for scheduling workings hours for each week for an annualised hours contract. ORTEC support all kind of companies with problems by analysing data and providing optimisation models.

We have for each employee a given contract with a minimum, maximum working hours and contract hours per week. Now, we determine the number of working hours of each employee for each week week. Further, we have a given demand in each week and we want the working hours and demand as close as possible. Besides this, we want that the difference between employees working hours and contract hours to be minimal. Therefore, our optimisation question is:

> *For which working schedule is the maximum over and under staffing minimal and the maximum difference between working hours and contract hours also minimal?*

2 Literature review

In this thesis, we make a model for an annualised hours problem. Therefore, in this section, we discuss literature concerning annualised hours models and heuristics used to solve such a model.

There are three different types of regulating working hours. You can measure the overtime, annualised hours or Working Time Accounts (WTAs). In a contract with overtime, a standard amount of working hours is formulated and an employee gets paid extra for overtime. In a contract with annualised hours, the working hours are measured over a longer time period, which for example can lead to a different amount of working hours each week. In a contract with WTA, an employee has a standard working hours, where an employee can be debited or credited to the standard working hours. Therefore, when an employee works more than his contract hours, these hours are credited to the WTA. In Corominas et al. [10], WTAs are considered, where the costs for capacity shortage, credited hours and overtime are minimised. In ull Hasan et al. [18], overtime is considered, here the total costs are minimised and the overtime is seen as added costs. Annualised hours give a constraint on the total contract hours per employee. These have to be equal to the sum of the contract hours over the whole period. This is discussed in van der Veen et al. [19].
In this thesis we consider annualised hours contract with minimum and maximum bound per week.

In the literature different types of objective functions. Corominas and Pastor [7, 8], ull Hasan et al. [18], van der Veen et al. [19] all considered the minimisation of the total costs, where in Corominas and Pastor [7, 8] additionally the total over and under staffing is minimised. Note that, although the objective function is different, these models have similar goals. When minimising the costs, the model also minimises the overtime, which correlates with a minimum over and under staffing. In Corominas and Pastor [7, 8], ull Hasan et al. [18], van der Veen et al. [19], there are constraints that state that the demand has to be met. This is done by hiring temporary workers. When the demand is not met, this is equivalent to a penalty. In our model, we combine these aspects and do not consider the total costs, but instead minimise the over and under staffing. Additionally, we minimise the difference between the working hours and contract hours.

In most annualised hours models, a deterministic demand is used, i.e., the demand is known for the whole period. However, Lusa et al. [15] uses a stochastic demand. It analyses a multistage problem, which has to make different decisions in each stage. This can be seen as a decision tree problem and gives different outcomes for each decision. Corominas and Pastor [8, 9] considers 3 types of demand; seasonal demand without a peak, seasonal demand with one peak and seasonal demand with two peaks.
In Corominas and Pastor [7, 8, 9], Corominas et al. [10], Lusa et al. [16], ull Hasan et al. [18], van der Veen et al. [19], multiple types of contracts are considered. There is made a difference between full time employees and temporary employees, whom can be hired when the demand can not be met. In van der Veen et al. [19], there is a scenario considered with hiring and firing. In Corominas and Pastor [7, 9], Corominas et al. [10], Lusa et al. [16], ull Hasan et al. [18], van der Veen et al. [19], there are different kind of skills, where for each employee is stated whether he is qualified for a specific skill. Different skills lead to extra constraints concerning whether an employee is qualified for a task. They also lead to different categories of demand for each task, which has to be taken into account.
In this thesis, we use a deterministic demand without specified peaks, with full-time and part-time contracts

with annualised hours. Furthermore, we consider a given amount of subcontractors and can not higher temporary employees. We do not consider different kind of skills. Therefore, there is only one type of demand per week.

Moreover, we want to take the computation complexity into account. In ull Hasan et al. [18], the LINGO solver is used. In Corominas and Pastor [7], a CPLEX solver was used. They choose a period of one year with different number of employees (10, 25, 50, 75, 100, 150, 200, 250). The lowest computation time was measured for 10 employees with 0.27 seconds and the highest for 250 employees with 58.22 seconds. A CPLEX solver was also used in Corominas and Pastor [8, 9], Lusa et al. [15], van der Veen et al. [19]. The computation time varies per model. However, in most cases it was measured around or below the 10 seconds with a higher computation time for bigger data sets. In Corominas and Pastor [8], the biggest data set that was used contained 5000 employees over a period of 46 weeks, with a highest computation time of 66.81 seconds.

Besides the models mentioned above, we consider a number of different used heuristics for solving annualised hours problems.
In Burke et al. [5], a nurse scheduling problem is described. There are 4 shifts a day and each shifts has to be assigned to a number of nurses. Different heuristics are used in this article. After creating an initial solution with heuristic ordering, a combination of *variable neighbourhood search*, heuristically unassigning shifts and repairing schedules using *heuristic ordering*. The article describes a variable neighbourhood search with the use of two neighbourhoods; the first is defined by assigning a shift to a different nurse, the second by swapping the nurses assigned to each of a pair of shifts. The second used heuristic removes all the nurses with the highest penalty and use heuristic ordering to reassigning the removed nurses. Subsequently, the variable neighbourhood search is used again to determine whether a better solution can be found.
In Attia et al. [2, 3], a job scheduling problem is represented, with multiple tasks, multiple skills (with efficiency per skill) and multiple employees. This model contains different constraints. These constraints guarantee that an employee can only be scheduled for one task for one skill a day and only if he is qualified to perform a task. Other constraints include; workload satisfaction, regulating working time, per day, week, 12 weeks and a year and constraints indicating the sequencing relationships between tasks.
In Attia et al. [3], there is a *genetic algorithm* used to solve this model. Here, first an initial solution is created with heuristic ordering. This is considered the initial population. The next generation is based on four groups, where parents are a combination of a task, employee and working time. From this, children are created. The first group are the parents with the lowest penalty, i.e., the elitist selection. The second group is created by selecting two parents and creating a child with an uniform crossover. One parent is selected from the elitist selection and one is chosen randomly. The third group is freshly generated and has no correlation with the initial population. The last group is a randomly selected group of parents. By using this approach, the algorithm aims to improve each generation.
In Attia et al. [2], a *greedy algorithm* is used. The tasks are categorised in most important/ most hard to fulfil. In each stage the employee with the highest efficiency and most availability is chosen for a task. In the article a comparison is made of the greedy algorithm with the genetic algorithm. When a genetic algorithm takes 155 seconds to compute the greedy algorithm only takes 1 second, for this problem.
In Dowsland [11], a nursing scheduling problem is considered, where the day and night shifts for each nurses has to be scheduled. The objective function is a minimisation of the total penalty of the working pattern for each nurse. The penalty encloses a mix of early and late shifts, preferred days off and the quality of a shift. After generating an initial solution, a combination of *local search* and *tabu search* is used. The local search exists out of different neighbourhoods. One neighbourhood exists out of different working patterns, another out of shifts and the last out of different nurses. Moves that are made in the last two neighbourhoods are not necessarily given a lower objective. In combination with tabu search, this causes that a search does not stop at a local optimum. Tabu search stores a small list of the previous found solutions and checks whether there are no cycles created while finding a better solution.
In this thesis, we use two different heuristics. One, the Lagrangian relaxation to reduce the computation time. Secondly, a hill climbing algorithm with local search with different neighbourhoods. The different neighbourhood are inspired by Dowsland [11], where one neighbourhood is of the employees and another of different weeks.

3 Background

In this chapter we discuss all the required background knowledge, which is needed for understanding the model and heuristics discussed in Chapter 4 and Chapter 5.

3.1. Computational complexity

When considering an optimisation problem or decision problem, you can define the computational complexity of this problem. The computational complexity is expressed in terms of the input and tells something about the running time of a model. This is expressed in terms of input, because the running time can differ by the software that is used. Therefore, we define the time complexity as follows: The *running time* or *time complexity* of a decision problem is formulated as the function $f : \mathbb{N} \rightarrow \mathbb{N}$, where *f(n)* is the maximum number of steps that decision problem needs to solve any input of length *n*. [17]

When considering the computational complexity it is important to make a difference between deterministic and nondeterministic algorithms. With a *deterministic* algorithm it is beforehand clear which path an algorithm follows. A *nondeterministic* algorithm can behave different in different runs depending on the choices that it makes. Because of this choices the algorithm can run in polynomial or exponential time. For example, this can occur in an algorithm with a random number generator.

To make a further distinction between algorithm we define the following complexity classes: *The class P* is the set of decision problems that is solvable in polynomial bounded running time.
The *class NP* is the set of all decision problems verifiable in polynomial bounded time.[20]
Solvable means that an solution is given on the problem and *verifiable* that for a problem a given solution can be verified.
Here P stands for deterministic Polynomial solvable decision problems and NP for Nondeterministic Polynomial decision problems. It is desirable to have a problem in class P, because then we can solve the problem in polynomial time and have control over the problem. One of the greatest unsolved questions in complexity theory is whether P equals NP. If this is true, then for every problem that is verifiable in polynomial time, there is an algorithm that solves the problem in polynomial time. What we do know for sure is that every problem that is solvable in polynomial time, is also verifiable in polynomial time.

A decision problem X is *NP-complete* if it satisfies two conditions:
1. X is in NP, and
2. every problem in NP is polynomial time reducible to X. [17]

A decision problem X is *NP-hard* if for some NP-complete problem Y there is a polynomial-time reduction from Y to X. [20]
Note, that a NP-hard problem is not solvable in polynomial time, unless P is equal to NP. To give an overview of the computational complexity definitions, see Figure 3.1 [12].

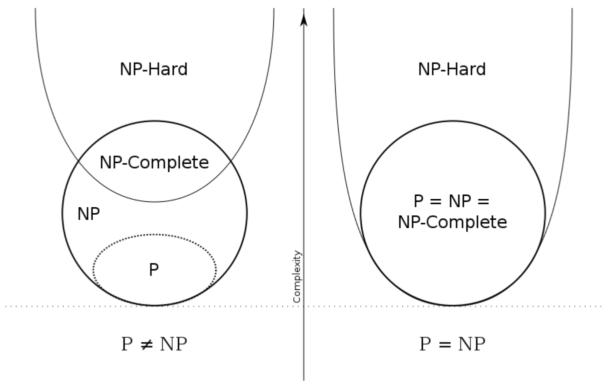

Figure 3.1: Euler diagram of P, NP, NP-complete and NP-hard

3.2. Mixed integer linear programming (MILP)

In this section, we introduce mixed integer linear programming. This is a model by which a decision problem is described. Such a model can be used for production planning, scheduling, networks and more.

Integer linear programming is a model with a linear objective function and linear constraints. Here the objective function is minimised or maximised. The objective function is a function of one or more variables, of which the values are unknown. The standard form of ILP is formulated as follows:

$$\begin{aligned} \max\ & \mathbf{c}^{\mathrm{T}}\mathbf{x} && (3.1)\\ \text{s.t.}\ & A\mathbf{x} \le \mathbf{b}, && (3.2)\\ & \mathbf{x} \ge \mathbf{0}, && (3.3)\\ & \mathbf{x} \in \mathbb{N}^n, && (3.4) \end{aligned}$$

In this formulation the parameters are $A, \mathbf{b}$ and $\mathbf{c}$. Here A is an $m \times n$-matrix, $\mathbf{b}$ an m-dimensional vector and $\mathbf{c}$ is as an n-dimensional vector. $\mathbf{c}^T\mathbf{x}$ is the objective function and $A\mathbf{x} \le \mathbf{b}$ are the constraints for the optimisation problem, where all the constraints are linear. The variables are represented by the vector $\mathbf{x}$, which is an n-dimensional vector that we have to optimise. For integer linear programming, the variables in the vector $\mathbf{x}$ have to be integers greater or equal to zero, as state in constraints (3.3) and (3.4).

For example a ILP problem can be formulated as follows:

$$\begin{aligned} \max\ & 5x - 2y && (3.5)\\ \text{s.t.}\ & x + 2y \le 6, && (3.6)\\ & 3x + 2y \le 8, && (3.7)\\ & x, y \ge 0, && (3.8)\\ & x \in \mathbb{N} && (3.9) \end{aligned}$$

This problem is visualised in Figure 3.2 and we later show how such a problem can be solved. A *feasible solution* is a solution for the optimisation problem for which all the constraints are met. In our example this are (3.6)-(3.9) and in Figure 3.2 the feasible region to the integer linear program is shown by the black points.

Mixed integer linear programming is a ILP problem, wherein variables can be integer as well as non-integer.

3.2.1. Cutting-plane method

The *Cutting-plane method* is an exact method to solve an MILP. It first makes use of LP relaxation to make the mixed integer linear program a linear program (LP), i.e., it removes the integer constraint. When there exists a feasible solution to a linear program, there is an extreme point where an optimal solution can be found. In Figure 3.2 the extreme points are the corner points, i.e., the intersections of the lines/ axis. The idea behind the cutting plane method is to add cuts to the linear program until an integer solution is obtained. A *cut* is a linear constraint which removes a part of the feasible region without cutting of the feasible integer solution. [4]

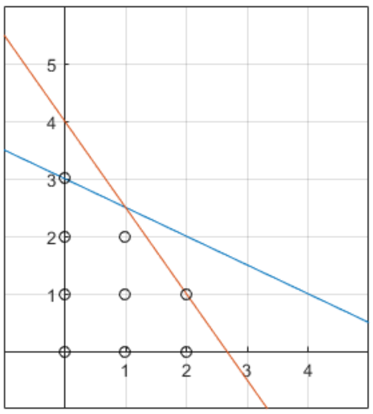

Figure 3.2: Feasible region of an ILP problem

3.2.2. Branch and bound

Branch and bound is another exact method for solving an MILP. The main idea behind branch and bound is to divide the feasible region in subdivisions, i.e., *branching* and when necessary again partitioning this subdivisions. The subdivisions are tested on *bounds*. For a maximisation problem we start with an upper bound of the solution to the LP problem. Here for, in the same way as the cutting-plane method, we first use LP relaxation to make a linear program (LP). When an integer solution is found, this integer solution becomes the lower bound. With each partition the method checks if the solution is in between the bounds and if a better bound is found. A subdivisions can not be further partitioned when or an integer solution is found or it is infeasible. [4]
Example: Consider the ILP problem mentioned in (3.5)-(3.9). In this example the optimal solution for the LP problem is $x = 2\frac{1}{2}$ and $y = 0$, with objective function value $5 \cdot 2\frac{1}{2} - 2 \cdot 0 = 12\frac{1}{2}$. Therefore, the upper bound for the optimum of the integer linear program is 12 (Because the variables are integer, the solution has to be integer as well). The next step is to partition the solution of x in $x \leq 2$ and $x \geq 3$. For $x \leq 2$ the optimum becomes $x = 2, y = 0$ and objective function value is 10, this is the best found feasible solution for the ILP and therefore becomes the lower bound. For $x \geq 3$ the model is infeasible. We now can not further partition the problem, because on one side an integer solution is found and the other side is infeasible. Therefore, the optimum for the ILP is 10.

3.2.3. Linearisation of quadratic problem

When formulating a problem, a quadratic constraint can be needed. Only this makes the problem not linear any more. Therefore, we introduce a linearisation technique with McCormick envelopes.
McCormick envelopes is a method that gives convex over and under estimators. This is done by introducing a substitute variable for the variable that is squared. With the addition of new constraints, we give a lower and upper bound for the quadratic variable. Using standard McCormick envelopes the additions to a model can be formulated as follows:

$$
\begin{aligned}
&x, y - \text{variables}\\
&x^L, x^U, y^L, y^U - \text{lower and upper bound of } x \text{ and } y\\
&w - \text{substitute variable representing } x \cdot y\\
\\
&w \geq x^L y + x y^L - x^L y^L\\
&w \geq x^U y + x y^U - x^U y^U\\
&w \leq x^U y + x y^L - x^U y^L\\
&w \leq x y^U + x^L y - x^L y^U\\
&x^L \geq x \geq x^U, y^L \geq y \geq y^U, w^L \geq w \geq w^U
\end{aligned}
$$

This can be made more precise with piecewise relaxation. Here, you make smaller enevelopes causing a closer estimate of the variable. A standard McCormick enevlope is depicted in Figure 3.3. The blue and the red lines are a representation of the constraints formulated before. [6]

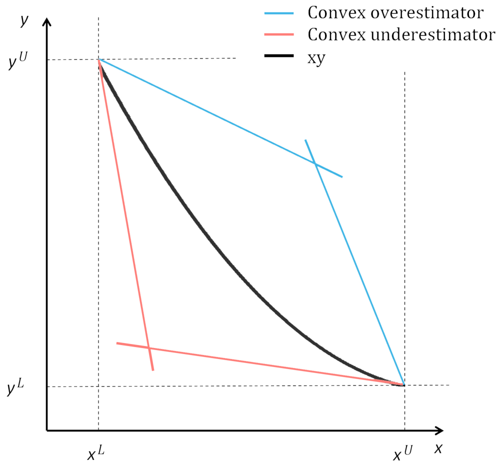

Figure 3.3: Standard McCormick envelope

3.3. Heuristic algorithm

There are different ways to solve an optimisation problem. An exact algorithm gives an optimal solution, but when coping with a NP-hard problem they have as disadvantage that there is no control over the computation time.

Therefore, we also consider heuristic algorithms. *Heuristic algorithms* are algorithms that use a smart way of searching for an optimal solution. There are many types of heuristic algorithms, we only discuss the algorithms we later use in this thesis. Some heuristics have the property that they can find a good solution in a given time or iterations. This gives us control over the computation time, but maybe not an optimal solution.

3.3.1. Lagrangian relaxation

Lagrangian relaxation is a heuristic method to solve complex problems. It removes difficult constraints from the original problem, to more efficiently find an optimal solution. It still takes these constraints into account in the formulation of the objective function, but these are added costs instead of constraints. It uses a non-negative multiplier as penalty if the removed constraint is violated. For this new problem an optimal solution is found. If a feasible solution exists, a large enough multiplier ensures that the found optimum also for feasible in the original model.

3.3.2. Local search

Local search is a method that starts with an candidate solution after which it moves to neighbour solution. It keeps moving from candidate solution to candidate solution, until a given bound is reached or a given number of iterations or time has passed. The neighbourhood for each problem can be differently described. In most cases a *neighbourhood* of a candidate solution is solution where a small change is made in the current solution. Local search methods are a good example of approximate algorithms; The method gives a solution at any given time or given number of iteration. This solution however does not have to be an optimal solution. The disadvantage of a local search method can be that the method visits the same solution multiple times and can because of this get stuck at a local optimum. [13]

3.3.3. Hill climbing

Hill climbing is a local search method. It starts with an initial solution after which it performs local search and checks if the new objective function value is better or not then the current objective function value. When a better feasible solution is found the current objective function value and current solution are updated. From this point, the method again tries to find a better feasible solution. This is repeated until a given number of iterations are time has passed. [13]

4

Model formulation

4.1. Mixed integer linear programming model

We want to find for each employee the number of working hours for each week. Our goal here is to minimise the maximum over and under staffing, while also minimising the maximum difference between the working hours and contract hours for each employee each week. We consider a couple of restrictions for this problem concerning the number of shifts an employee, the length of a shift and restrictions concerning the contract of an employee. Therefore, we first state the following sets, parameters and variables:

Sets:	
I	Employees
T	Time period in weeks
Parameters:	
$\|T\|$	Total number of weeks in set$T$
d_t	Demand in week $t \in T$ (in hours)
c_i	Contract hours per week for employee $i \in I$
lc_i, uc_i	Minimum and maximum working hours per week for employee $i \in I$
ls_0, us_0	Minimum and maximum hours per shift
ls_k, us_k	Minimum and maximum shifts for an employee in $k \in \mathbb{N}$ weeks
$a_{it} \in \{0,1\}$	0 if employee $i \in I$ is absent in week $t \in T$,, 1 otherwise
$f_i \in \{0,1\}$	1 if employee $i \in I$ has a full-time contract, 0 otherwise
Variables:	
Y_{it}	Number of hours that employee $i \in I$ works in week $t \in T$
$V_{it} \in \mathbb{N}$	The number of shifts for employee $i \in I$ in week $t \in T$

We formulate our model based on the model formulated in van der Veen et al. [19]. This model can be found in Appendix A. The signifanct difference between the model described in [19] and in this thesis is that in [19] different skills are considered and the objective function exists out of the total costs. Besides this, we not only consider working hours, but also working shifts.

Model 1.1

In the first model, we define the maximum over and under staffing as a quadratic function, in which the more the total working hours differ from the demand, the greater the penalty. We do the same for the difference between the working hours and contract hours. This formulated in objective function (4.1), which has two parts. First, we minimise the difference between the total working hours and contract hours per week, $(Y_{it} - c_i \cdot a_{it})$. Note that we only measure the difference if an employee works and therefore multiply c_i with a_{it}. Here is a_{it} for employee i in week t 1 if he is available and 0 if he is absent. Secondly, the over/under staffing is formulated by ($\sum Y_{it} - d_t$). Parameters λ_1 and λ_2 determine the importance of each component.

We further have constraints (4.2)-(4.6). Constraints (4.2) state that an employee works between the minimum and maximum contract hours (lc_i, uc_i) and an employee does not work if he is absent (a_{it}). Constraints (4.3), state the minimum and maximum hours in a shift (ls_0, us_0). Constraints (4.4) give a restriction on the minimum and maximum number of shifts, for a given number of weeks. This could be one week or a longer period. To state these constraints, we first translate the number of hours per week given by Y_{it} to the number of shifts, V_{it}, for employee i in week t.

The last constraints (4.5) ensure that the number of hours worked in the total time period, $\sum_{t\in T} Y_{it}$, correspond to the annualised contract hours (the sum of the contract hours over the whole period), $|T| \cdot c_i$. The total of number of weeks over the whole period is indicated with $|T|$. Further, we state if an employee is absent, that he gets payed for his contract hours. Constraints (4.6) state that an employee must have a non-negative integer number of working hours Y_{it} and number of shifts V_{it}.

$$\min \ \lambda_1 \cdot \sum_{i\in I} \sum_{t\in T} (Y_{it} - c_i \cdot a_{it})^2 + \lambda_2 \cdot \sum_{t\in T} (\sum_{i\in I} Y_{it} - d_t)^2 \tag{4.1}$$

$$\text{s.t.} \ lc_i \cdot a_{it} \le Y_{it} \le uc_i \cdot a_{it}, \qquad \forall i \in I, \forall t \in T \tag{4.2}$$

$$V_{it} \cdot ls_0 \le Y_{it} \le V_{it} \cdot us_0, \qquad \forall i \in I, \forall t \in T \tag{4.3}$$

$$ls_k \cdot a_{it} \le \sum_{t=j}^{j+k-1} V_{it} \le us_k \cdot a_{it}, \qquad \forall i \in I, \forall t, j \in T, \forall k \in \mathbb{N} \tag{4.4}$$

$$\sum_{t\in T} Y_{it} + (1 - a_{it}) \cdot c_i = |T| \cdot c_i, \qquad \forall i \in I \tag{4.5}$$

$$Y_{it}, V_{it} \in \mathbb{N} \qquad \forall i \in I, \forall t \in T \tag{4.6}$$

To linearise the quadratic objective function, we could use McCormick envelopes as described in Chapter 3.

Model 2.1

The disadvantage of McCormick envelopes is that for a good estimation they give gives a considerably amount of extra constraints and complexity. Therefore, we consider another way to formulate the objective function. In the second model, we avoid a quadratic formulation by using the maximum function and formulate the objective function as follows:

$$\min \ \lambda_1 \cdot \sum_{i\in I} \max_{t\in T}(|Y_{it} - c_i \cdot a_{it}|) + \lambda_2 \cdot \max_{t\in T}(|\sum_{i\in I} Y_{it} - d_t|) \tag{4.7}$$

Where λ_1 and λ_2 are parameters which indicates the importance of each component.

However, in a linearised model we can not have absolute values or maximum functions. To linearise the absolute values we take the maximum of $Y_{it} - c_i \cdot a_{it}$ and $c_i \cdot a_{it} - Y_{it}$. We do the same with $\sum_{i\in I} Y_{it} - d_t$.
To linearise the maximum functions we introduce substitute values m_i and n with the following constraints:

$$m_i \ge Y_{it} - c_i \cdot a_{it} \qquad \forall t \in T \tag{4.8}$$

$$m_i \ge c_i \cdot a_{it} - Y_{it} \qquad \forall t \in T \tag{4.9}$$

$$n \ge \sum_{i\in I} Y_{it} - d_t \qquad \forall t \in T \tag{4.10}$$

$$n \ge d_t - \sum_{i\in I} Y_{it} \qquad \forall t \in T \tag{4.11}$$

Substituting m_i and n gives us the objective function:

$$\min \ \lambda_1 \cdot \sum_{i\in I} m_i + \lambda_2 \cdot n \tag{4.12}$$

Model 2.2

The objective function in Model 2.1 only consider the maximum difference between the contract hours and the working hours, maximum difference between the total contract hours and demand. Moreover, we want that the difference is minimised for each week. This is of less importance and has a lower weight (λ_3, λ_4). The objective function can now be formulated as following:

$$\min \ \lambda_1 \cdot \sum_{i\in I} m_i + \lambda_2 \cdot n + \lambda_3 \cdot \sum_{i\in I}\sum_{t\in T} (|Y_{it} - c_i \cdot a_{it}|) + \lambda_4 \cdot \sum_{t\in T} (|\sum_{i\in I} Y_{it} - d_t|) \tag{4.13}$$

We again linearise the objective function by introducing substitute variables p_{it} and q_t to replace the absolute values with the following constraints:

$$p_{it} \geq Y_{it} - c_i \cdot a_{it} \qquad \forall i \in I, \forall t \in T \tag{4.14}$$
$$p_{it} \geq c_i \cdot a_{it} - Y_{it} \qquad \forall i \in I, \forall t \in T \tag{4.15}$$
$$q_t \geq \sum_{i\in I} Y_{it} - d_t \qquad \forall t \in T \tag{4.16}$$
$$q_t \geq d_t - \sum_{i\in I} Y_{it} \qquad \forall t \in T \tag{4.17}$$

Substituting p_{it} and q_t results in the following objective function:

$$\min \ \lambda_1 \cdot \sum_{i\in I} m_i + \lambda_2 \cdot n + \lambda_3 \cdot \sum_{i\in I}\sum_{t\in T} p_{it} + \lambda_4 \cdot \sum_{t\in T} q_t \tag{4.18}$$

Model 3.1

We made the assumption that each contract is either full time or part time, but to expand the model we also take subcontractors into account. For a subcontractor the minimum and maximum working hours per week are equal to the contract hours per week. Therefore, we say that a subcontractor either works his contract hours in a given week or does not work at all. This means that the constraint for annualised hours does not apply to a subcontractor. For this, we introduce parameter f_i which is 1 if an employee has a full time or part time contract and 0 when an employee is a subcontractor. We also introduce a binary variable co_{it} which is 1 if employee i works in week t and is 0 if employee i does not work in week t. For a full time contract this is equal to a_{it} and for a subcontractor this depends if he works in a given week. This results in the final MILP model:

$$\min \ \lambda_1 \cdot \sum_{i\in I} m_i + \lambda_2 \cdot n + \lambda_3 \cdot \sum_{i\in I}\sum_{t\in T} p_{it} + \lambda_4 \cdot \sum_{t\in T} q_t \tag{4.19}$$

$$\text{s.t.} \ a_{it} \cdot f_i \leq co_{it} \leq a_{it} \qquad \forall i \in I, \forall t \in T \tag{4.20}$$
$$lc_i \cdot co_{it} \leq Y_{it} \leq uc_i \cdot co_{it}, \qquad \forall i \in I, \forall t \in T \tag{4.21}$$
$$V_{it} \cdot ls_0 \leq Y_{it} \leq V_{it} \cdot us_0, \qquad \forall i \in I, \forall t \in T \tag{4.22}$$
$$ls_k \cdot co_{it} \leq \sum_{t=j}^{j+k-1} V_{it} \leq us_k \cdot co_{it}, \qquad \forall i \in I, \forall t, j \in T, \forall k \in \mathbb{N} \tag{4.23}$$
$$f_i \cdot \big(\sum_{t\in T} Y_{it} + (1 - a_{it}) \cdot c_i\big) = f_i \cdot \big(|T| \cdot c_i\big), \qquad \forall i \in I \tag{4.24}$$
$$m_i \geq Y_{it} - c_i \cdot co_{it} \cdot f_i \qquad \forall t \in T \tag{4.25}$$
$$m_i \geq c_i \cdot co_{it} \cdot f_i - Y_{it} \qquad \forall t \in T \tag{4.26}$$
$$n \geq \sum_{i\in I} Y_{it} - d_t \qquad \forall t \in T \tag{4.27}$$
$$n \geq d_t - \sum_{i\in I} Y_{it} \qquad \forall t \in T \tag{4.28}$$
$$p_{it} \geq Y_{it} - c_i \cdot co_{it} \cdot f_i \qquad \forall i \in I, \forall t \in T \tag{4.29}$$
$$p_{it} \geq c_i \cdot co_{it} \cdot f_i - Y_{it} \qquad \forall i \in I, \forall t \in T \tag{4.30}$$
$$q_t \geq \sum_{i\in I} Y_{it} - d_t \qquad \forall t \in T \tag{4.31}$$
$$q_t \geq d_t - \sum_{i\in I} Y_{it} \qquad \forall t \in T \tag{4.32}$$
$$Y_{it}, V_{it}, m_i, n, p_{it}, q_t \in \mathbb{N} \qquad \forall i \in I, \forall t \in T \tag{4.33}$$
$$co_{it} \in \{1, 0\} \qquad \forall i \in I, \forall t \in T \tag{4.34}$$

Constraints (4.20) are added to the model, which causes that co_{it} depends on a_{it}. For this, we can replace a_{it} with co_{it}. Constraints (4.24) are multiplied with f_i, causing that annualised hours only apply for full time and part time employees.

5 Heuristic method

In Appendix B we give an attempt of proving the model given in Chapter 4.1 is NP-hard. This proof is still incomplete, but an interesting beginning. Because of NP-hardness heuristics can be preferred, because these can give more control over the computation time. Therefore, we consider in this chapter two types of heuristic algorithms. The first Lagrangian relaxation, which gives a lower computation time, but does not provide control over the computation time. Second we consider a hill climbing algorithm. This algorithm uses local search and updates in each iteration the best found solution.
Our goal is to find for each employee, the number of working hours per each week such that the maximum difference between the working hours and the contract hours and the maximum difference between the total working hours and demand per week are minimised.

5.1. Lagrangian relaxation

The reason for using Lagrangian relaxation is that difficult constraints become a penalty in the objective function. This causes that a feasible solution can be easier to find and when α is chosen high enough, the found optimal for Lagrangian relaxation is also feasible in the initial model.
Therefore, we remove constraints (4.24) and put a penalty in the objective function. We choose constraints (4.24), because this are the only constraints with a equality and therefore the hardest to full fill. The objective function is formulated as follows:

$$\min\ \lambda_1 \cdot \sum_{i\in I} m_i + \lambda_2 \cdot n + \lambda_3 \cdot \sum_{i\in I}\sum_{t\in T} p_{it} + \lambda_4 \cdot \sum_{t\in T} q_t + \alpha \cdot (f_i \cdot |\big(\sum_{t\in T}(Y_{it} + (|T| - a_{it}) \cdot c_i)\big) - |T| \cdot c_i| \tag{5.1}$$

We linearise this in the same way as we did for the linearisation of (4.13) and introduce substitute variable lag_i which we substitute in the objective function for $|\big(\sum_{t\in T}(Y_{it} + (|T| - a_{it}) \cdot c_i)\big) - |T| \cdot c_i|$ and the constraints given by:

$$lag_i \geq \big(\sum_{t\in T} Y_{it} + (|T| - a_{it}) \cdot c_i\big) - |T| \cdot c_i \qquad \forall i \in I \tag{5.2}$$

$$lag_i \geq |T| \cdot c_i - \big(\sum_{t\in T} Y_{it} + (|T| - a_{it}) \cdot c_i\big) \qquad \forall i \in I \tag{5.3}$$

We choose α significantly bigger then $\lambda_1, \lambda_2, \lambda_3, \lambda_4$. In this way if a feasible solution exists for the Lagrangian relaxation, it is also feasible in the initial model.

5.2. Hill climbing method

First we generate an initial solution. Therefore, we determine if an employee i works in week t, given by co_{it}. We do this by considering the availability of an employee and the demand. If employee i is absent in week t, $co_{it} = 0$. Further, we say subcontractors only work in week t when there is under staffing. We determine this by calculating $\sum_{i\in I} Y_{it} - d_t$ for each week.

Now we generate for each employee the working hours y_{it} by multiplying the contract hours c_i by co_{it}.

On this initial solution we use a hill climbing algorithm with two local search operations: $Move_{employee}$ and $Move_{week}$. Because the objective function exists out of two parts we consider two neighbourhoods, which are determined by $N_{contract}$ and N_{demand}.

$N_{contract}$ - provides employee $i_{contract}$ and week $t_{contract}$ for which $|Y_{it} - c_i|$ has the highest value

N_{demand} - provides week t_{max}, t_{min} for which $\sum_{i\in I} Y_{it} - d_t$ has the highest and lowest value

$Move_{employee}$ - For employee i it swaps a given amount of hours from week t_1 and t_2. Illustrated in Figure 5.1.

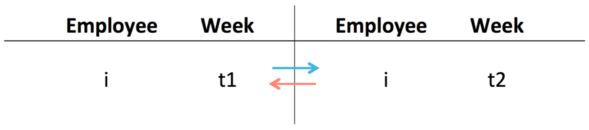

Figure 5.1: Move hours between employees

$Move_{week}$ - This operation moves hours from employee i_1 to i_2 for week a given week t_1. However this creates a violation in the constraints (4.24) (which concern annualised hours). To not violate this constraints we make an additional swap for each employee with a given week t_2. For example when two hours are subtracted from the working hours of employee i_1 in week t_1, then the additional the operation adds two hours to the working hours of employee i_1 in week t_2. The same is done for employee i_2. This is illustrated in Figure 5.2.

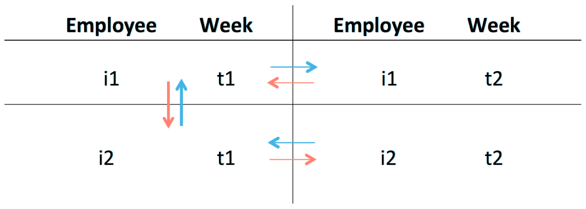

Figure 5.2: Move hours between weeks

For each iteration we perform two kind of moves between weeks and one move between employees. After each operation we check if the operation improves the current found objective function value. If this is the case we update the current solution and from here we again try to find a better solution.

The first $move_{employee}$ is based on improving $N_{contract}$. We say $i = i_{contract}$ and $t_1 = t_{contract}$. To determine t_2 we see if $Y_{it} - c_i$ is positive or negative. When positive we define $t_2 = t_{min}$ and move hours from t_1 to t_2. When negative we define $t_2 = t_{max}$ and move hours from t_2 to t_1. Where t_{max} and t_{min} are given by N_{demand}

Further, $move_{employee}$ chooses a random i and t_1 and then finds the t_2 where for the objective function after the swap is the lowest. This swap is to prevent that the algorithm stays in the same neighbourhood.

The $move_{week}$ operation is based on improving $N_{contract}$. We say $i_1 = i_{contract}$ and $t_1 = t_{contract}$. We choose i_2 and t_2 at random. When $Y_{it} - c_i$ is positive we move hours from (i_1, t_1) to (i_2, t_1) and when negative vice versa.

We repeat this steps until an given number of iterations or amount of time has passed.

6 Results MILP model

In this section, we discuss the performance of Model 3.1 described in Section 4.1.

6.1. Data and parameters

We test the model with generated data with a random instance generator provided by ORTEC. In this generator the number of employees, weeks and skills can be chosen. After this, a dataset is created with the following data:

- Contract working hours for each employee per week (c_i)
- Minimum and maximum working hours for each employee per week (lc_i, uc_i)
- Demand in hours per week (d_t)
- Weeks that each employee is absent (a_{it})
- If an employee is qualified for a given task

In the model described in this thesis, we have only one skill and have that all the employees are qualified for this skill.
In the generated data, there are three type of contracts: Full time contracts, part time contracts and subcontractors. Full time contracts and part time contracts, are provided with minimum, maximum and standard contract hours. Subcontractors have a contract where the minimum, maximum and contract hours are the equal to each other. In this data generator the contract hours for subcontractors are eight hours a week. For subcontractors, we state that they or work all their contract hours or do not work in a week.

Further, we specify the remaining parameters in our model. The duration of a shift of an employee is between 4 and 9 hours (ls_0, us_0), with a maximum of 6 shifts a week (us_1), a maximum of 11 shifts in two weeks (us_2), a maximum of 20 shifts in four weeks (us_4). The minimum number of shifts for a given period is set to 0 (ls_1, ls_2, ls_4). We set f_i to 0 if the minimum, maximum and contract hours are the same and to 1 otherwise.

We choose $\lambda_1, \ldots, \lambda_4$ in decuple of each other, with the more important a component in the objective is, the greater. We set $\lambda_1 = 10$, $\lambda_2 = 100$, $\lambda_3 = 1$ $\lambda_4 = 1$.

We aim to find a model for a spanning horizon of 52 weeks and 100 employees. However, we also test the model for data sets of 50, 150, 200, 250 and 1000 employees over 52 weeks. Addtionaly we test the data for 100 employees, where we change the weeks to 26 and 78 weeks.

6.2. Results

The model is implemented in Spyder (Python 3.6) with the program Annaconda. For solving the MILP, we use Gurobi 8.0. Gurobi is an advanced MIP-solver which uses different algorithms and heuristics. It has 17 different types of cutting plane methods and 14 different heuristics. We compare the minimum, average ($\bar{t}$), maximum and median ($\tilde{t}$) computation time of each data set, which is shown in Table 6.1 and Tabel 6.2. For each data set we toke 100 samples generated with the random data generator mentioned in Section 6.1.

Employees	t_{min}	$\bar{t}$	t_{max}	$\tilde{t}$
50	1.68	2.68	11.76	2.48
100	3.60	6.14	15.07	5.68
150	5.50	10.02	17.58	10.09
200	8.27	16.16	47.96	15.08
250	11.52	21.96	50.43	21.20
1000	92.15	253.97	1700.82	173.54

Table 6.1: Computation times (in seconds) for Gurobi for 52 weeks

Weeks	t_{min}	$\bar{t}$	t_{max}	$\tilde{t}$
26	1.81	2.81	4.49	2.74
52	3.60	6.13	15.07	5.68
78	5.50	11.33	42.53	10.40

Table 6.2: Computation times (in seconds) for Gurobi for 100 employees

In Table 6.1, Table 6.2 and Figure 6.1, is shown that for bigger data sets, and thus more variables, we find a longer computation time. Here we plotted the data of 1000 employees and 52 weeks apart, because of the much longer computation time. What is remarkable that for larger data sets the minimum and maximum are further apart from the average computation time. In Figure 6.1, is shown that there are a few data sets with a much longer computation time then the average. This causes that the median of the computation time is in most cases lower then the average computation time, shown in Table 6.1 and Table 6.2.

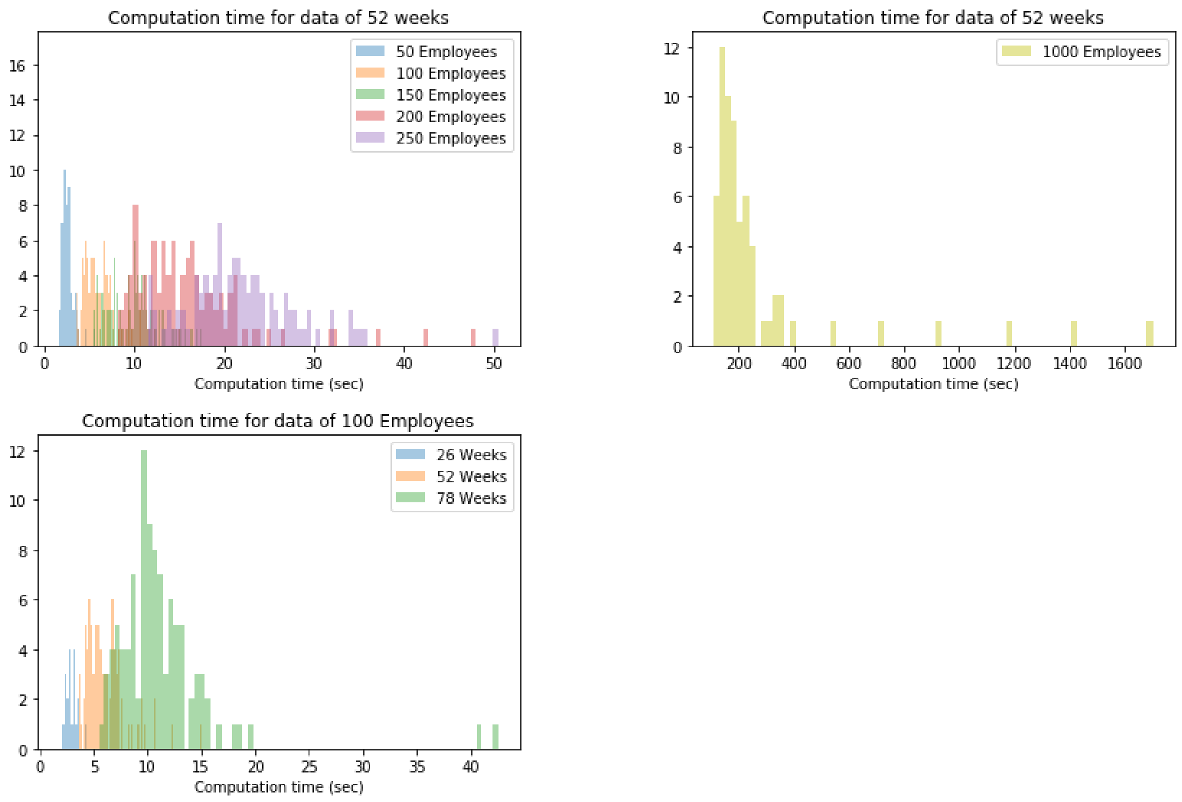

Figure 6.1: Computation time (in seconds) with for different data

In Figure 6.2, we compare data sets of the same size. Starting with 100 employees and 52 weeks, we halved / doubled either the weeks or the employees. It is clearly shown that the computation time for both are practically the same. Therefore, we can conclude that only the size of the data has influence on the computation time and it does not matter if this is because of more weeks or more employees.

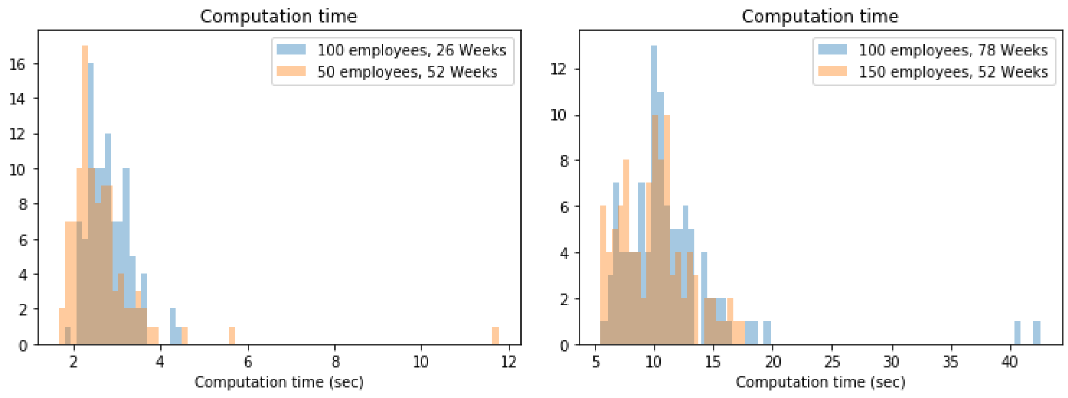

Figure 6.2: Computation time (in seconds) with for same data size

Last, we plot the computation time for different components of the data to find if there is a correlation. For this we take the data set of 100 employees and 52 weeks and 1000 employees and 52 weeks. We choose these two because of there size difference and expect that they give similar correlations. In Figure 6.3 and Figure 6.4, a scatter plot of both are shown. In each scatter plot, the linear trend is shown. We see a positive linear trend for the correlation with the objective function value / total working hours / total demand and a negative linear trend for the absolute difference between the total working hours and total demand.

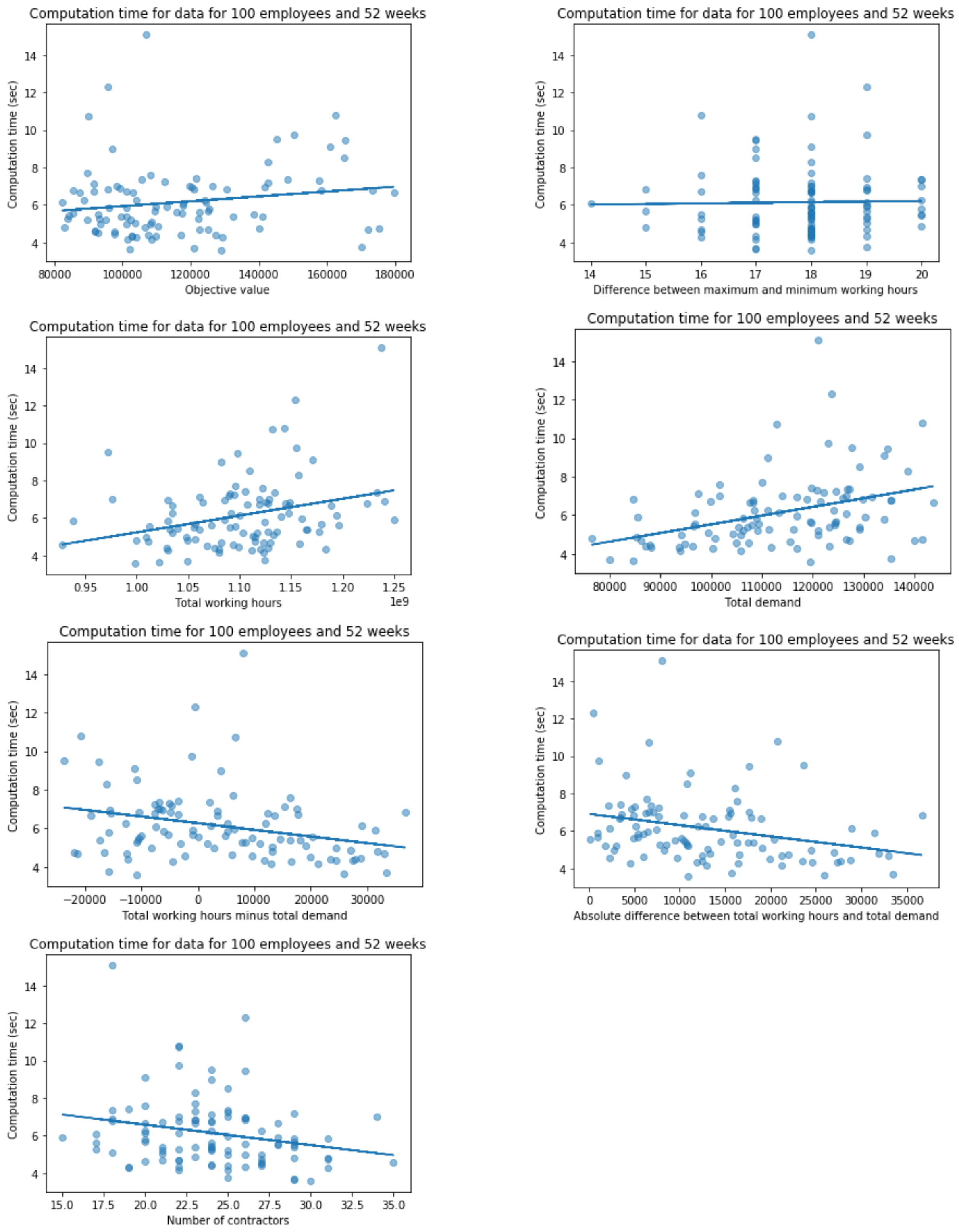

Figure 6.3: Computation time (in s) with dataset of 100 employees and 52 weeks

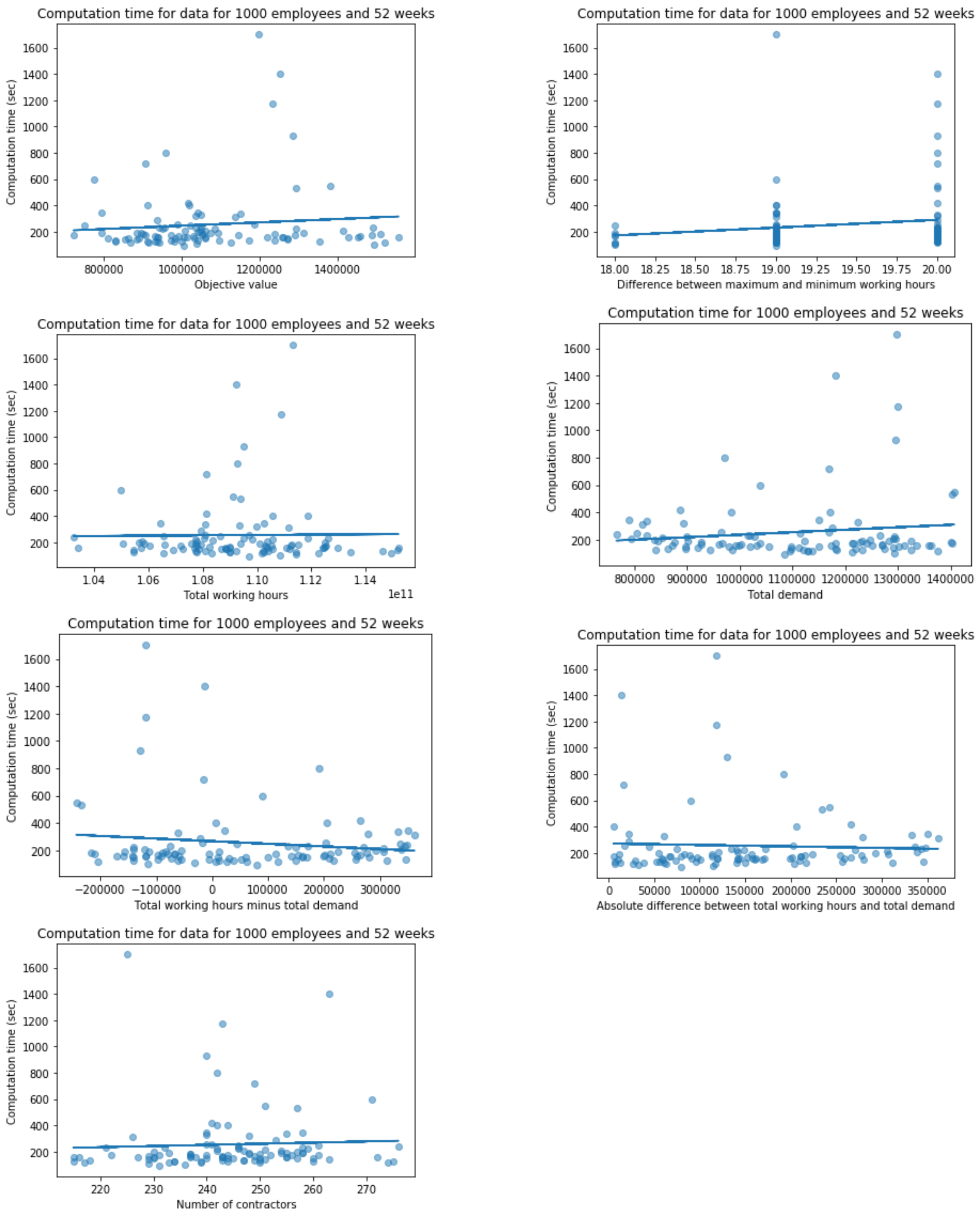

Figure 6.4: Computation time (in s) with dataset of 1000 employees and 52 weeks

7

Comparison with heuristic algorithm

In this chapter we discuss the performance of the Lagrangian relaxation and hill climbing method.

7.1. Lagrangian relaxation

With Lagrangian relaxation, we put the constraint for annualised hours in the objective function. We set $\alpha = 1000$, which is ten times larger than the value of $\lambda_1, \ldots, \lambda_4$ in the MILP model. In this way, we create a big enough penalty for which a feasible solution for the initial problem is generated. Note that this only happens when a feasible solution to the initial problem exists. We again compare the computation time of different data and give the minimum, average ($\bar{t}$), maximum and median ($\tilde{t}$) computation time of each data set. This is shown in Table 7.1.

Employees*	t_{min}	$\bar{t}$	t_{max}	$\tilde{t}$
50	1.87	2.93	8.25	2.72
100	3.96	6.06	12.91	5.63
150	5.30	9.03	29.64	8.21
200	8.46	14.25	39.92	11.68
250	10.88	19.74	50.43	18.06
1000	71.22	220.61	1711.33	153.48
Weeks**	t_{min}	$\bar{t}$	t_{max}	$\tilde{t}$
26	2.28	3.18	7.68	2.92
52	3.96	6.06	12.91	5.63
78	6.71	10.48	57.39	8.78

Table 7.1: Computation times (in seconds) for Lagrangian relaxation
* with 52 weeks
** with 100 empployees

Comparing Table 6.1 and Table 6.2 with Table 7.1, we find lower computation time for the model with langrange relaxation. In Figure 7.2 this difference is clearly shown. The largest difference between the average computation time is for a data set with 1000 employees and 52 weeks. Although the average computation time is lower for the model with Lagrangian relaxation, this does not guarantee a lower computation time. For example the maximum computation time for 150 employees and 52 weeks is longer for the Lagrangian relaxation as can be seen in Figure 7.2.

7.2. Hill climbing method

We use 50 iterations in the hill climbing method described in Section 5.2. This is because in the first 50 iterations the biggest improvements in the objective function value are made. After 50 iterations, often a higher objective function value is found and thus often no improvement. Comparing Table 6.1 and Table 6.2 with Table 7.2, we find that the Hill climbing method has significantly lower computation time. Figure 7.2

shows that the hill climbing model can best be used for large data sets. In small data sets it not always gives a better computation time.

Employees*	t_{min}	$\overline{t}$	t_{max}	$\tilde{t}$
50	2.72	3.42	4.60	3.42
100	5.26	6.37	8.24	6.44
150	8.26	10.08	14.26	10.05
200	10.53	13.09	15.53	13.30
250	13.48	16.41	21.92	16.49
1000	61.11	76.25	99.86	75.19
Weeks**	t_{min}	$\overline{t}$	t_{max}	$\tilde{t}$
26	2.87	3.51	5.61	3.45
52	3.96	6.06	12.91	5.63
78	7.58	9.75	14.45	9.74

Table 7.2: Computation time (in seconds) of hill climbing method
* with 52 weeks
** with 100 empployees

In Table 7.3 we evaluate the quality of the found objective function value by the Hill climbing method. We do this by calculating for each data set γ, where γ is given by:

$$\gamma = \frac{\gamma^{\bullet} - \gamma^{*}}{\gamma^{*}}$$

with $\gamma^{\bullet}$ the found objective function value by the hill climbing method and γ^{*} the objective function value from the exact method. In Table 7.3, we show the minimum γ, average ($\overline{\gamma}$) and maximum γ. Here, we see that $\overline{\gamma}$ is higher for larger data sets. Further, we see that $\overline{\gamma}$ is always above 0.1, i.e., the objective function value with hill climbing is more than 10 percent higher than the optimal objective function value. In Figure 7.1, gamma is shown for the different data.

Employees*	γ_{min}	$\overline{\gamma}$	γ_{max}
50	0.026	0.128	0.260
100	0.058	0.152	0.261
150	0.080	0.166	0.283
200	0.097	0.176	0.257
250	0.106	0.192	0.270
1000	0.133	0.209	0.291
Weeks**	γ_{min}	$\overline{\gamma}$	γ_{max}
26	0.066	0.163	0.310
52	0.058	0.152	0.261
78	0.056	0.146	0.263

Table 7.3: Difference exact method and Hill climbing method
* with 52 weeks
** with 100 empployees

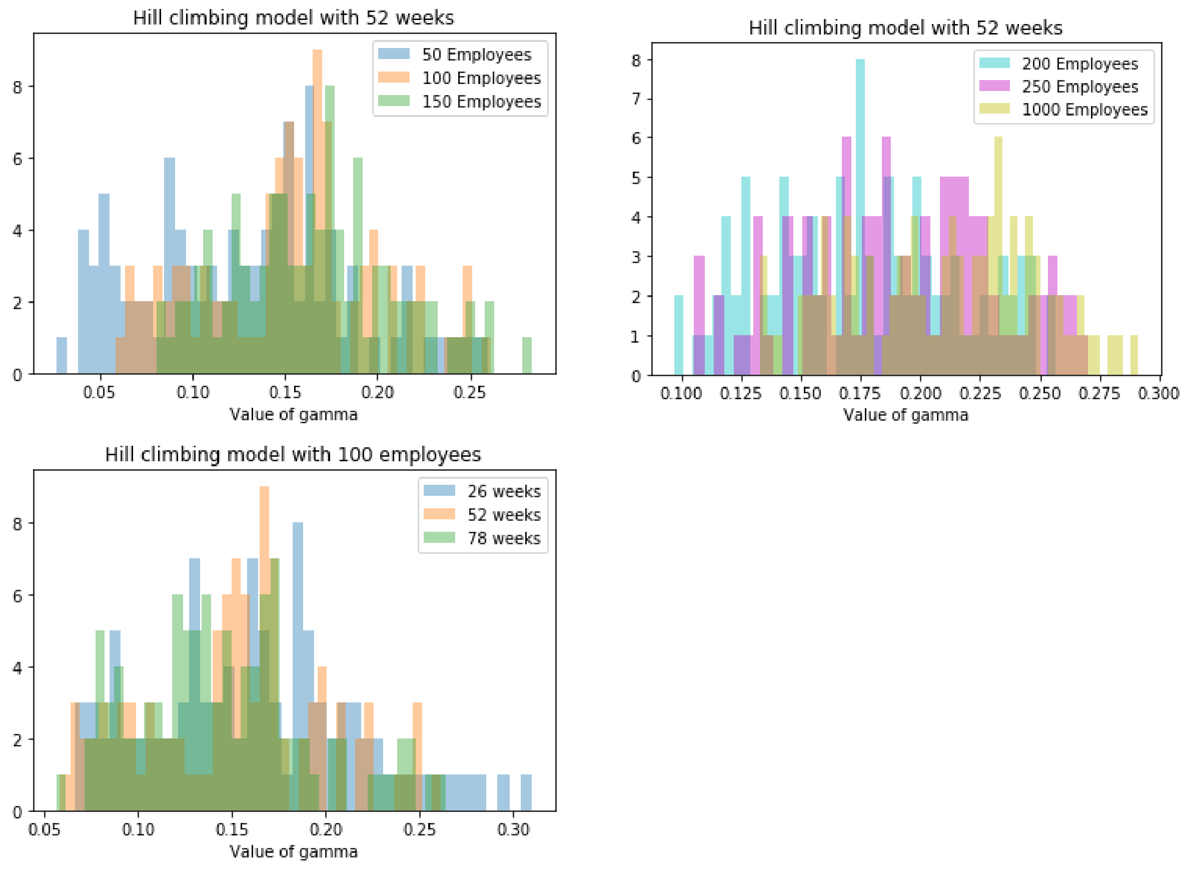

Figure 7.1: Comparison γ for different data

7.3. Gurobi with intial solution

When solving the model with an exact algorithm, it could take some time before the algorithm finds a feasible solution. Therefore, we test if the computation time improves if we start the exact Gurobi algorithm with a feasible solution. This feasible solution is generated with the hill climbing method with 50 iterations. However, as shown in Table 7.4 and Figure 7.3 the computation time does not improves. This can be because Gurboi it self uses better heuristics for finding a good first feasible solution.

Employees*	t_{min}	$\overline{t}$	t_{max}	$\tilde{t}$
50	2.40	4.08	10.56	3.88
100	5.99	9.72	22.27	9.17
150	9.30	16.47	25.73	15.76
200	14.66	26.48	50.12	24.96
250	19.16	36.94	65.37	35.52
1000	199.68	394.85	1432.68	302.21
Weeks**	t_{min}	$\overline{t}$	t_{max}	$\tilde{t}$
26	2.83	3.86	7.84	3.70
52	5.99	9.72	22.27	9.17
78	12.07	19.40	48.90	17.72

Table 7.4: Computation time (sec) of Gurobi with initial solution
* with 52 weeks
** with 100 empployees

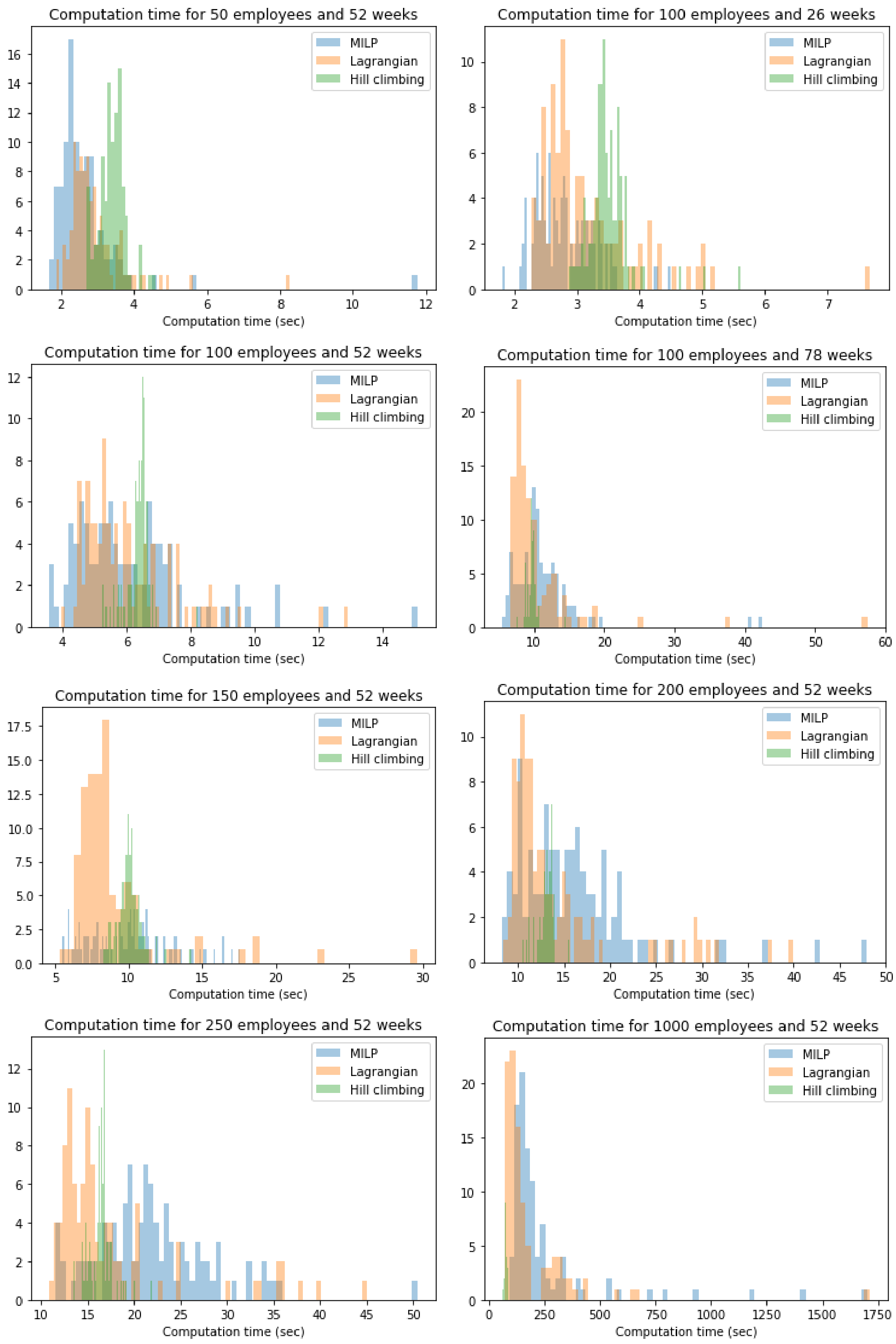

Figure 7.2: Comparison computation time (in s) for different data

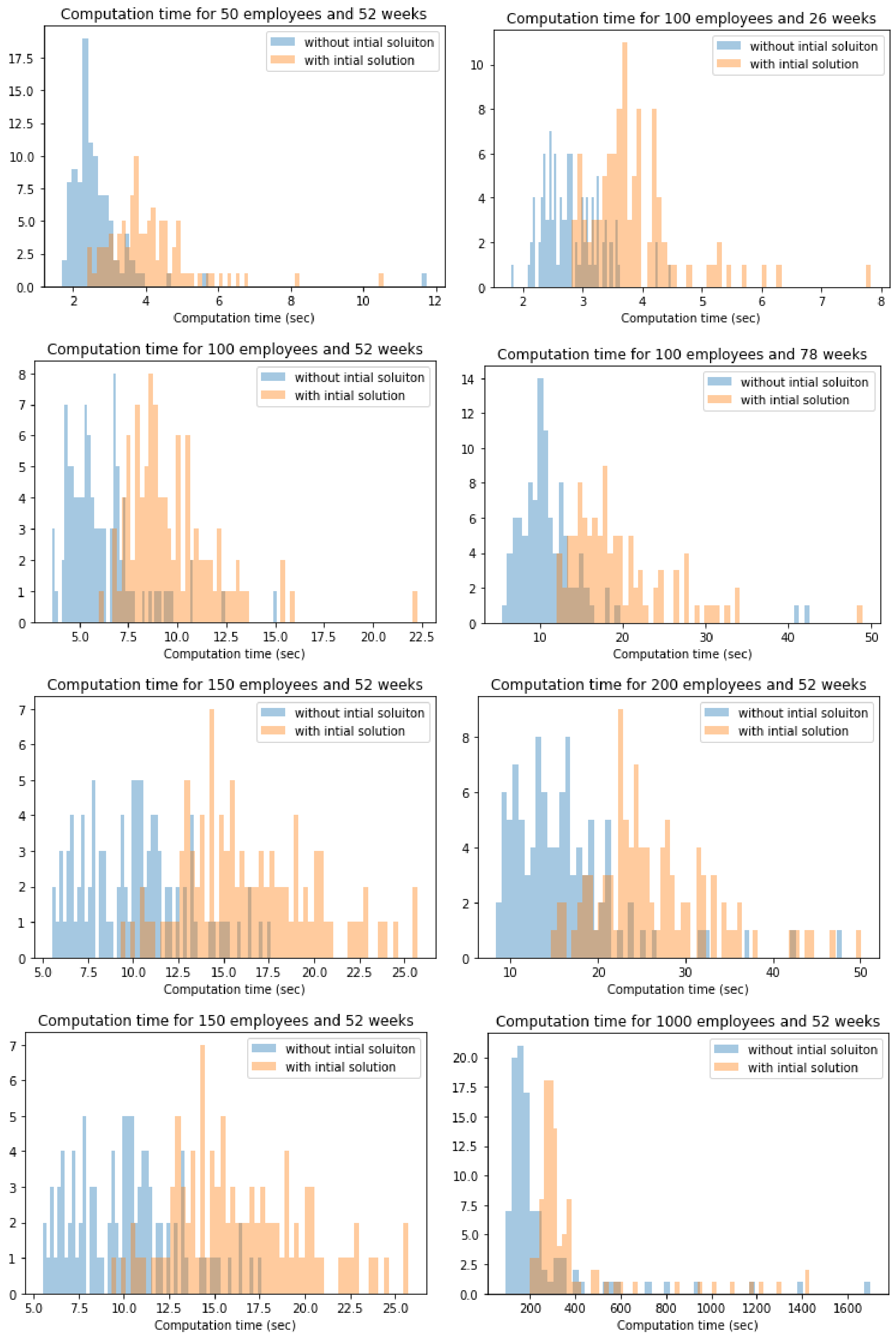

Figure 7.3: Comparison computation time (in s) for different data

8 Conclusion, discussion and recommendation

In this thesis, our goal was to find a model, which gives a working schedule for 100 employees and 52 weeks, for which the maximum over and under staffing is minimal and the maximum difference between working hours and contract hours minimal as well. We have formulated this in an mixed integer linear program and have given the beginning of a proof that states the model is NP-hard. Although, the proof is incomplete, we still consider this an important part of the thesis. In further research we would recommend to improve the proof of NP-hardness.

We have found that a Gurobi solver solves the model in a reasonable computation time with an average of 6 seconds for 100 employees and 52 weeks. In Figure 7.2, it was clearly shown that there is a longer computation time for larger data sets.
For Lagrangian relaxation, we have found a better computation time than for Gurobi. However, the difference for 100 employees and 52 weeks is hardly notable and the effect of Lagrangian relaxation is more distinguishable for bigger data sets. We have found similar correlations between the size of the data and the computation time; for larger data sets, there was a longer computation time.
The hill climbing algorithm gave a consequent and low computation time. The objective function value, however, differs considerably and in most cases it gave a difference of more than 10 percent.
Last we combined the found solutions of the hill climbing method and set this as start solution in the Gurobi algorithm. This, however, gave no improvements and even lead to a longer computation time.

Altogether, we can conclude that the Lagrangian relaxation can best be used, as this gives the best results; an exact solution and had a reasonable computation time. The only disadvantage of this algorithm is, that there is no control over the computation time. In further research the heuristics could be studied to a greater extend, so that these will, firstly, give a better approximation of the optimal solution and, secondly, a better control over the computation time.

Bibliography

[1] E. Appelbaum, T. Bailey, P. Berg, and A. L. Kalleberg. Shared work-valued care: New norms for organizing market work and unpaid care work. 2001.

[2] E. Attia, P. Duquenne, and J. Le-Lann. A greedy heuristic approach for the project scheduling with labour allocation problem. *12th Al-Azhar Engineering International Conference (AEIC2012)*, pages 25–27, 2012.

[3] E. Attia, P. Duquenne, and J. Le-Lann. Considering skills evolutions in multi-skilled workforce allocation with flexible working hours. *International Journal of Production Research*, 52(15):4548–4573, 2014.

[4] S. Bradley, H. Hax, and T. Magnanti. *Applied mathematical programming*. Addison Wesley, 1977.

[5] E. K. Burke, T. Curtios, G. Post, R. Qu, and B. Veltman. A hybrid heuristic ordering and variable neighbourhood search for the nurse rostering problem. *European Journal of Operational Research*, 188(2): 330–341, 2008.

[6] P. Castro. *A Tighter Piecewise McCormick Relaxation for Bilinear Problems*. 2014.

[7] A. Corominas and A. Lusa R. Pastor. Using milp to plan annualised working hours. *Journal of the Operational Research Society*, 53(10):1101–1108, 2002.

[8] A. Corominas and A. Lusa R. Pastor. Planning annualised hours with a finite set of weekly working hours and joint holidays. *Annals of Operations Research*, 128:217, 2004.

[9] A. Corominas and R. Pastor. Replanning working time under annualised working hours. *International Journal of Production Research*, 48(5):1493–1515, 2010.

[10] A. Corominas, J. Olivella, and R. Pastor. Capacity planning with working time accounts in services. *Journal of the Operational Research Society*, 61:321, 2010.

[11] K.A. Dowsland. Nurse scheduling with tabu search and strategic oscillation. *European Journal of Operational Research*, 106:393–407, 1998.

[12] B. Esfahbod. P vs np problem, 2007. URL `https://commons.wikimedia.org/wiki/File:P_np_np-complete_np-hard.svg`.

[13] H. H. Hoos and T. Stützle. *Stochastic Local Search: Foundations and Applications*. 2001.

[14] S.J.C. Joosten. Relaxations of the 3-partition problem. Master's thesis, Department of Applied Mathematics, University of Twente, 2011.

[15] A. Lusa, A. Corominas, and N. Munoz. A multistage scenario optimisation procedure to plan annualised working hours under demand uncertainty. *Int J prod Econ*, 113(2):957–968, 2008.

[16] A. Lusa, A. Corominas, and R. Pastor. An exact procedure for planning holidays and working time under annualized hours considering cross-trained workers with different efficiencies. *International Journal of Production Research*, 50(6):1568–1581, 2008.

[17] M. Sipser. Introduction to the theory of computation. *Boston, United States of America: Thomson Course Technology*, 2:247–283, 2006.

[18] M.G. ull Hasan, I. Ali, and S.S. Hasan. Annualized hours planning with fuzzy demand constraint. *ProbStat Forum*, 6:50–56, 2016.

[19] E. van der Veen, E.W. Hans, B. Veltman, L.M. Berrevoets, and H.J.J.M. Berden. A case study of cost-efficient staffing under annualized hours. *Health Care Management Science*, 18:279–288, 2015.

[20] J. van Leeuwen. *Algorithms and Complexity. Handbook of Theoretical Computer Science, Vol. A.* Elsevier Science, Amsterdam, 1998.

A
Model from E. van der Veen et al.

In this appendix, we describe the MILP model used in van der Veen et al. [19].

Sets:	
I	Set of employees, indexed by i
J	Set of skills, indexed by j
$J_i \subset J$	Set of skills of employee i
T	Set of time slots, indexed by t
S	Set of subsets of T, indexed by s
$T_s \subset T$	Subsets s of time slots for which working hours constraints have to be enforced
Parameters:	
d_{jt}	Demand for skill j in time slot t (in hours)
c_i^{fix}	Fixed cost of employee i
c_i^{var}	Variable cost of employee i
l_{it}, u_{it}	Minimum and maximum working hours of employee i in time slot t
l_i^s, u_i^s	Minimum and maximum total working hours of employee i in time slots of T_s
l_i, u_i	Minimum and maximum total working hours of employee i during the entire planning horizon T
Variables:	
X_{ijt}	Number of hours employee i works on skill j during time slot t
Y_i	1 if employee i is selected in the workforce, 0 if not
TC	Total cost of all employees
TC^{fix}	Sum of fixed cost of all employees
TC^{var}	Sum of variable cost of all employees

In (A.1) the objective function is formulated as the total costs.
In constraints (A.2) we make sure that the demand is met in every time slot for every skill, i.e., under staffing is not aloud. Constraints (A.3) ensure that in every time slot, the working hours of every employee are between the lower and the upper bound and multiplying with Y_i enforces that when an employee is not selected for the workforce he does not work. For every $s \in S$, there is a similar lower and upper bounds on the total working time, stated in constraints (A.4). Constraints (A.4) give the opportunity to model for example constraints on the minimum and maximum working time in 4 or 13 week periods. In constraints (A.5) is stated that for every employee the working hours are between the lower and upper bound of the planning horizon. The fixed and variable cost are defined by constraint (A.6) and (A.7).
As last is given in constraints (A.8) and (A.9) that the working hours of an employee are non negative integers and in (A.9) that Y_i takes only binary values.

$$\min \quad TC = TC^{fix} + TC^{var} \tag{A.1}$$

$$\text{s.t.} \quad \sum_{i \in I} X_{ijt} \geq d_{jt} \qquad \forall j \in j, \forall t \in T \tag{A.2}$$

$$l_{it} Y_i \leq \sum_{j \in J_i} X_{ijt} \leq u_{it} Y_i \qquad \forall i \in I, \forall t \in T \tag{A.3}$$

$$l_i^s Y_i \leq \sum_{t \in T_s} \sum_{j \in J_i} X_{ijt} \leq u_i^s Y_i \qquad \forall i \in I, \forall s \in S \tag{A.4}$$

$$l_i Y_i \leq \sum_{t \in T} \sum_{j \in J_i} X_{ijt} \leq u_i Y_i \qquad \forall i \in I \tag{A.5}$$

$$TC^{fix} = \sum_{i \in I} c_i^{fix} Y_i \tag{A.6}$$

$$TC^{var} = \sum_{i \in I} c_i^{var} \big(\big(\sum_{t \in T} \sum_{j \in J_i} X_i jt \big) - l_i Y_i \big) \tag{A.7}$$

$$X_{ijt} \geq 0 \qquad \forall i \in I, \forall j \in J, \forall t \in T \tag{A.8}$$

$$Y_i \in \{1, 0\} \qquad \forall i \in i \tag{A.9}$$

$$X_{ijt} \in \mathbb{N} \qquad \forall i \in I, \forall j \in J, \forall t \in T \tag{A.10}$$

B Proof model in this thesis is NP-hard

We prove that the model described in Section 4.1 is NP-hard by reducing the 3-partition problem to this problem. We cite from Joosten [14]:"*The 3-partition problem is a problem where one has to partition 3q numbers(allowing duplicates) into q groups of 3, such that each group has the same sum.*". These numbers are all positive integers.
It is a well-known fact that the 3-partition problem is NP hard. We consider our model with only three employee, which gives us $3t$ working hours to divided in groups of three. We set the parameters $\lambda_1 = \lambda_3 = \lambda_4 = 0$, $\lambda_2 = 1$, $f_i = 1$, $a_{it} = 1$, $ls_0 = 0$ and $us_0 = us_k = \infty$. We choose $k = 0$, which results in that (4.23) can be left out. Because $\lambda_1 = \lambda_3 = \lambda_4 = 0$ constraints (4.25)-(4.26),(4.29)-(4.32) can be omitted. Moreover, we choose $d_t = 0$ which makes constraints (4.27) and (4.28) the same. The model now can be formulated as follows:

$$\min n \tag{B.1}$$

$$\text{s.t. } 1 \le co_{it} \le 1 \qquad \forall i \in I, \forall t \in T \tag{B.2}$$

$$lc_i \cdot co_t \le Y_{it} \le uc_i \cdot co_{it}, \qquad \forall i \in I, \forall t \in T \tag{B.3}$$

$$0 \le Y_{it} \le \infty, \qquad \forall i \in I, \forall t \in T \tag{B.4}$$

$$\sum_{t \in T} Y_{it} = |T| \cdot c_i \qquad \forall i \in I \tag{B.5}$$

$$n \ge \sum_{i \in I} Y_{it} \qquad \forall t \in T \tag{B.6}$$

$$V_{it}, Y_{it}, co_{it} \in \mathbb{N} \qquad \forall i \in I, \forall t \in T \tag{B.7}$$

Constraint (B.2) state that $co_{it} = 1$. We substitute this in the model. Further, constraints (B.4) and (B.6) state that Y_t can take any value between 0 and ∞ causing that these constraints can be omitted. This results in the following model:

$$\min n \tag{B.8}$$

$$\text{s.t. } lc_i \le Y_{it} \le uc_i, \qquad \forall i \in I, \forall t \in T \tag{B.9}$$

$$\sum_{t \in T} Y_{it} = |T| \cdot c_i \qquad \forall i \in I \tag{B.10}$$

$$n \ge \sum_{i \in I} Y_{it} \qquad \forall t \in T \tag{B.11}$$

$$Y_{it} \in \mathbb{N} \qquad \forall i \in I, \forall t \in T \tag{B.12}$$

We consider each week as a group of three numbers, Y_{it}. Constraint (B.11) and the objective function state that the minimisation of this sum. This minimisation causes that the sum for each week is as low as possible and thus equal. The only problem in this prove is that the working hours Y_{it} are variable and not a given set of numbers.
Additionally, consider a set of $3t$ positive integers, which can be divided in t groups three with a equal sum. The sum of such a group is equal to n and thus described by constraint (B.11). Further, c_i is chosen in such a way that constraints (B.10) hold $\forall i \in I$ and thus describes the model from Section 4.1.

If these comparisons where complete, this would have proven that the model described in Section 4.1 is NP-hard. However, we still miss that a set of Y_{it} is given beforehand and constraints (B.10) can not be included completely.

C
Python code

In this appendix, the python code is given, which is used for solving the initial MILP problem. This is done in python with Gurobi.

```
from gurobipy import *
import time

#data inlezen
Data = "data1"
Input = open('1000E52W'+Data + '.txt', 'r')

DataAll = []
for line in Input:
    DataAll.append(line)
Input.close()
print(Input)

DataC = []
dt = []
for line in DataAll:
    line = line.replace(';',' ')
    line = line.split()
    if len(line) > 0:
        if str(line[0]) == 'true':
            DataC.append(line)
    if len(line) == 1:
        try:
            dt.append(int(line[0]))
        except:
            0

li = []
ci = []
ui = []
for line in DataC:
    line[1] = line[1].replace(',',' ')
    line[1] = line[1].split()
    li.append(int(line[1][0]))
    ci.append(int(line[1][1]))
    ui.append(int(line[1][2]))
```

```
ni = int(len(ci))
nt = int(len(dt))
A1 = []
for line2 in DataC:

    if len(line2) < 3:
        A1.append([])
    else:
        line = line2[2]
        line = line.replace('\n','')
        line = line.replace(',',' ')

        line = line.split()
        Aux = []
        for letter in line:
            Aux.append(int(letter))

        A1.append(Aux)

ait = tuplelist()
for i in range(ni):
    ai = []
    for j in range(nt):
        if j+1 in A1[i]:
            ai.append(0)
        else:
            ai.append(1)
    ait.append(ai)

fi = []
for i in range(ni):
    if ci[i] == li[i] and ci[i] == ui[i]:
        fi.append(0)
    else:
        fi.append(1)

start_time = time.time()
#parameters
l1 = 10
l2 = 100
l3 = 0.1
l4 = 0.2

#shift parameters
ls0 = 4
us0 = 9
ls1= 0
us1= 6
ls2= 0
us2 = 11
ls4 = 0
us4 = 20

#milp model
try:
    m = Model("MILP")
```

```
#variables ni employees, ntweeks
vit = m.addVars(ni , nt , vtype = GRB.INTEGER, name ="shifts")
coit = m.addVars(ni,nt,vtype=GRB.BINARY, name = "coit")
yit = m.addVars(ni , nt , vtype = GRB.INTEGER, name ="workinghours")
pit = m.addVars(ni , nt , vtype = GRB.INTEGER, name ="pit")
mi = m.addVars(ni , vtype = GRB.INTEGER, name = "mi")
qt = m.addVars(nt , vtype = GRB.INTEGER, name ="qt")
n = m.addVar(lb = 0,ub = GRB.INFINITY, vtype = GRB.INTEGER, name = "n")


#Set Objective
m.setObjective( l1 * quicksum(mi[i] for i in range(ni)) + l2 * n + l3 * quicksum(pit[i,t] for i in

m.Params.MIPGap = 10**-5
#Add Constraint :lci * fi < coit < ait
for i in range(ni):
    for t in range(nt):
        m.addConstr((fi[i]*ait[i][t] <= coit[i,t]), "coit0")
        m.addConstr((ait[i][t] >= coit[i,t]), "coit1")
#Add Constraint :lci * ait < yit < uci *ait
for i in range(ni):
    for t in range(nt):
        m.addConstr((li[i]*coit[i,t] <= yit[i,t]), "c0")
        m.addConstr((ui[i]*coit[i,t] >= yit[i,t]), "c1")
m.update()

for i in range(ni):
    for t in range(nt):
        m.addConstr((ls0*vit[i,t] <= yit[i,t]), "shifts uren lb")
        m.addConstr((us0*vit[i,t] >= yit[i,t]), "shifts uren ub")
m.update()

for i in range(ni):
    for t in range(nt):
        m.addConstr(ls1 *coit[i,t] <= vit[i,t], "shifts per week lb")
        m.addConstr(us1 >= vit[i,t], "shifts per week ub")
m.update()

for i in range(ni):
    for t in range(nt-1):
        m.addConstr((us2 >= quicksum(vit[i,j] for j in [t,t+1])), "shifts 2 weken ub")
        m.addConstr((ls2 <= quicksum(vit[i,j] for j in [t,t+1])), "shifts 2 weken ub")
m.update()

for i in range(ni):
    for t in range(nt-3):
        m.addConstr((us4 >= quicksum(vit[i,j] for j in [t,t+1,t+2,t+3])), "shifts 4 weken ub")
        m.addConstr((ls4 <= quicksum(vit[i,j] for j in [t,t+1,t+2,t+3])), "shifts 4 weken ub")
m.update()

for i in range(ni):
    m.addConstr(fi[i]*quicksum(yit[i,t] - ait[i][t]*ci[i] + ci[i] for t in range(nt)) == fi[i]*ci[i
m.update()

for i in range(ni):
        for t in range(nt):
            m.addConstr(mi[i] >= (yit[i,t] - ci[i]*coit[i,t]), "m1")
            m.addConstr(mi[i] >= (ci[i]*coit[i,t] - yit[i,t]), "m2")
    m.optimize ()

    for t in range(nt):
        m.addConstr(n >= yit.sum('*',t) - dt[t], "n1")
        m.addConstr(n >= dt[t]- yit.sum('*',t), "n2")
    m.optimize ()

    for i in range(ni):
        for t in range(nt):
            m.addConstr(pit[i,t] >= (yit[i,t] - ci[i]*coit[i,t]), "p1")
            m.addConstr(pit[i,t] >= (ci[i]*coit[i,t] - yit[i,t]), "p2")
    m.optimize ()

    for t in range(nt):
        m.addConstr(qt[t] >= yit.sum('*',t) - dt[t], "q1")
        m.addConstr(qt[t] >= dt[t]- yit.sum('*',t), "q2")
    m.optimize ()


    for v in m.getVars():
        print('%s %g' % (v.varName, v.x))
    print('Obj: %g' % m.objVal )
except GurobiError as e:
    print('Error code' + str(e. errno ) + ": " + str(e))
except AttributeError :
    print('Encountered an attribute error')
Objec =  str(m.objVal)
Time = str(time.time() - start_time)
print("%s" % (time.time() - start_time))

open("MILP250E52W.txt", "a").write('\n')
open("MILP250E52W.txt", "a").write(Data + ' ,'+ Objec + ' ,' + Time)
```