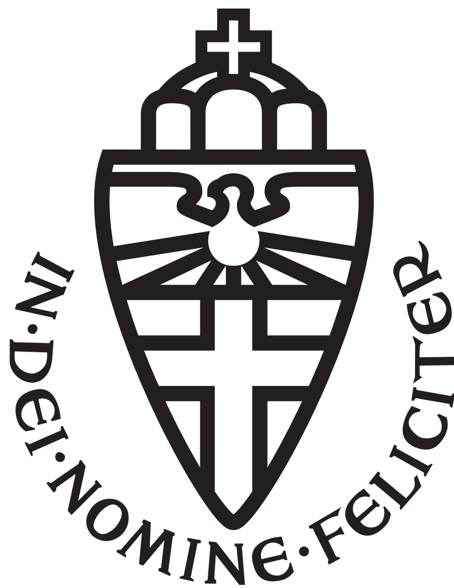
Self-rostering with employee preferences
Master thesis

Frank van Hoof, s4369513

February 24, 2021

Supervisors: Wieb Bosma and Elwin Jongeling



Faculty of Science
Department of Mathematics
Radboud University Nijmegen

Contents

Preface **5**

1 Introduction **6**

2 Scope **8**
2.1 Working regulations . . . 8
2.2 Employment conditions . . . 8
2.3 Preferences . . . 9
2.4 Skills . . . 10

3 Linear programming **11**
3.1 Introduction . . . 11
3.1.1 Integer linear programming . . . 14
3.2 History . . . 14
3.3 Algorithms . . . 16
3.3.1 Fourier-Motzkin . . . 16
3.3.2 Simplex . . . 17
3.3.3 Ellipsoid method . . . 17
3.3.4 Branch-and-bound . . . 18

4 Creating possible schedules **19**
4.1 Category of schedules . . . 19
4.2 Overview model . . . 20
4.3 Possible schedules . . . 20
4.3.1 Reducing the number of shifts . . . 21
4.3.2 Algorithm . . . 21
4.3.3 Holidays . . . 22
4.3.4 Removing duplicates . . . 24
4.3.5 Variations . . . 24

5 Finding a roster **26**
5.1 Schedule value and kudos . . . 26
5.1.1 Calculating the value of a schedule . . . 27
5.1.2 Kudos . . . 27
5.1.3 Personalised value . . . 28
5.1.4 Updating kudos . . . 29
5.2 Linear programming model . . . 32
5.3 CPLEX . . . 35
5.3.1 Improvements . . . 35

6 Tests **39**
6.1 Possible schedules . . . 39
6.2 Kudos . . . 41
6.2.1 Does it select preferences? . . . 42
6.2.2 Prefer to have a day off on the same day . . . 42
6.2.3 Prefer to work on the same day . . . 43
6.2.4 Concerning employee without preferences . . . 44

6.2.5 Prefer the same day off/work revisited 45
6.2.6 Conflict with three employees 46
6.2.7 Behaviour with different contract hours 48
6.2.8 Lots of conflicting preferences 50
6.3 Model 54
6.3.1 16 employees 54
6.3.2 48 employees 56

7 Conclusions **59**
7.1 Possible schedules 59
7.2 Kudos 59
7.2.1 Creating $c_{s,e}$ 59
7.2.2 Recalculating kudos 60
7.2.3 Scaling kudos 60
7.2.4 Average kudos 61
7.3 Model 61

8 Future research **62**

Appendices **64**

A Possible schedules results **64**

B Kudos results **65**
B.1 Prefer to have a day off on the same day 65
B.2 Prefer the same day off revisited 67
B.3 Conflict with three employees 69
B.4 Behaviour with different contract hours 73

C Model results **76**
C.1 Test with 16 employees 76
C.2 Test with 48 employees 78

Preface

This document is a report of my master thesis and wouldn't have been possible without the help of others. While working on this thesis I was supervised by Wieb Bosma, employee at the Radboud University Nijmegen, and Elwin Jongeling, employee at Ordina NL. On behalf of Ordina I researched the possibilities on considering preferences when creating rosters for employees.

I'd like to thank both Wieb Bosma and Elwin Jongeling for working with me, helping me and giving their honest opinions and suggestions. Both of your input helped me a lot while researching and writing.

My thanks also goes out to the other employees at Ordina. Not only for helping me out with the thesis, but also for being welcoming. I have enjoyed the time with everyone.

Last but not least I want to thank my family and friends. Not only for supporting me during the last year, but for all those years I've been studying mathematics. I'd like to mention my father in particular as he also took the time to read my thesis and give some input.

I hope everyone reading this will read it with as much enthusiasm and joy as I had while researching this subject.

1 Introduction

The Supply Chain Optimization, SCO, department of Ordina deals with all sorts of optimization problems such as rostering and routeplanning. The problem at hand is a self-rostering problem and it is usually solved using linear programming. The roots of linear programming lie in the 19th century and the subject is still actively studied to this day. More information on its history can be found in Section 3.2. SCO would like an improved model that will take preferences into account to increase the satisfaction of the employees. This is inspired by an earlier project; however there currently isn't a specific customer.

To increase the satisfaction of employees, companies want them to be able to have some say in their working hours. Not only can planners then swap shifts to create a better roster, but also have some insight in what the preferences of employees are and, where possible, adjust when shifts start and end. However all the data used is fictional. For this reason an analysis of the preferences and a possible adjustment in shifts is purposeless. The focus will be on making sure everyone is satisfied without changing shifts. To do this, we will have to treat every employee equal as much as possible, independent of factors such as skills or days off.

As mentioned, this problem has been considered before with a former customer of SCO, and one important conclusion was that the way of rostering requires some form of transparency. It is important that an employee knows why he didn't get the shift he preferred while a colleague did get his preference. So it is also a goal to be able to explain why the final roster is fairest, and more importantly, that differences are noticed and will be remembered. Because if we can find the fairest solution, but can't show the employees that we considered everyone's preferences, we won't obtain the increase in satisfaction that we want. A struggle that many others also had, see [1], [2].

Although the preferences of employees is our focus in this thesis, the over- and understaffing of shifts isn't allowed if it can be avoided. I.e, it is considered worse than an employee not getting his preference.
There may be some employees with skills and some shifts that require those skills. As a second constraint it is not allowed for employees that lack the required skills to be assigned to these shifts.

The speed of finding a roster isn't the main concern. The focus will lie on the validity of the roster and fairness to the employees. Validity of the roster doesn't just concern over- and understaffing and skills, but also taking into account some working regulations. More details on the working regulations that will be considered are mentioned in Section 2.1.

So, summarized, the goals are:

- increase satisfaction of employees;
- transparency to the employees;
- create no over- or understaffing;
- only assign employees that have the required skills;
- working regulations must be satisfied.

The rosters that are created will have a planning horizon of one week. It is in most cases impossible to create a roster for one week in which all employees are given equally fair schedules. Therefore

kudos will be introduced. These will be used to keep track of the number of preferred shifts an employee has been assigned, compared to other employees. Kudos will be increased if an employee is given a schedule which is worse than the schedules other employees got, and decreased if it's better.

The employees are placed in categories to find the schedules faster. Employees that are comparable in factors such as contract hours, days off and preferences, will be placed in the same category. After doing this we will find the possible schedules for employees and value those depending on their preferences and current amount of kudos. Then these possible schedules and their values are used to find the fairest selection of schedules to create a valid roster. Finally the kudos of employees will be updated accordingly so they can be used for creating next week's roster.

These kudos have two main advantages. First of all, it is possible to determine who has been given schedules that contained more preferences compared to other employees, and to use this to even out the score in upcoming weeks.
As a second advantage, these kudos can be published for transparency. It will show employees that have been given a worse schedule compared to other employees, that this has been noticed by an increase in their amount of kudos and that this will be taken into consideration when creating the next roster.

2 Scope

In reality there are a lot of factors that can be considered while making a roster and they widely depend on the company. Some only have a few fixed shifts, others have so many small differing activities that you will need a new shift starting for every hour, or even more. Some companies may require 24/7 coverage while others never work during the night or the weekends. Because we will not consider an actual case it is important to determine which factors we want to look at. To keep it fairly realistic, we want to take into account some of the more general factors. Throughout the thesis we will require a **24/7 coverage** where the demand on a given moment will vary depending on factors such as nights and weekends.

2.1 Working regulations

As mentioned before, we will consider a one week **planning horizon**, Monday through Sunday. Some of the working regulations are spread out over a year. There is, for example, a maximum number of night shifts that can be assigned per year. Because the roster will span only one week, it is hard to take this rule into account. Instead there will be a possibility to either work **night shifts** or don't work any night shifts for each employee separately.
Another rule that will not be considered is the **breaks during a long shift**. To simplify the model we will assume that these breaks are included in the shifts already.
However, there are some working regulations that will be considered. First, there is the **weekend**. It will be mandatory for each employee to have at least two days off in a row, at least once per week. To determine this, we will not consider the previous and next week. This means that if an employee has a day off on Monday and works on Tuesday, that this will not be considered a weekend even though in theory the Sunday prior to this week could also have been a day off. The reason to do this is that there currently is no data of previous weeks and nothing about the Sunday prior to this week is known.
Another rule that will be considered is the **break between two shifts**. After each shift there must be at least a 9 hour break, and no two shifts can be started on the same day, even if there is more than a 9 hour break between them. The **length of the shifts** will always be 8 hours.

2.2 Employment conditions

Contract hours can vary a lot between employees and the amount of variance will differ per company. However most of the time there will be a distinction between part time and full time contracts. To keep it relatively simple we will consider only those two cases: full time consisting of 38 hours per week and part time 20 hours. An employee must never work more or less than a **5 hour interval around his contract hours**. This, together with the shifts that are all 8 hours in length, means that and employee working full time will always have 5 shifts per week where a part time employee will have either 2 or 3 shifts per week.
Employees might have **days off** that are determined beforehand and must never be violated.
Finally we will also consider **holidays**. These are days that employees requested leave, these must also not be violated. Holidays are always full days. So holidays and days off are very similar. Their difference is the fact that having a day off may not violate the contract hours margin whereas the holiday may. A holiday can be seen as dropping a shift and thus reducing the working hours.

2.3 Preferences

In reality preferences can vary a lot. For example an employee could prefer to be unavailable for a few hours in the morning to bring his children to school, which would then only be preferred during weekdays. An other may want an entire day off. A third might also prefer to be unavailable the first few hours in the morning, because he prefers to sleep longer. Or someone could have a preference to work on a praticular day.
So there are a lot of ways to set preferences. Also one might want to set some form of priority to those preferences, to make a distinction between someone who brings their children to school and someone who prefers to sleep longer. In this thesis the priority of preferences will not be considered. However the method suggested does still support a lot of freedom and can easily be adjusted to do so.

Trying to keep the **possibilities for preferences** fairly large, there are a few different ways to assign them.
An employee can set a preference to work or to be free at a certain time. The possible time periods that can be selected are either a full day or a part of a day.
The days are split into four parts and depend on the starting time of the shift:

- night shifts start between 21:00 and 1:59;
- morning shifts start between 2:00 and 7:59;
- day shifts start between 8:00 and 12:59;
- evening shifts start between 13:00 and 20:59.

It is important to note that an employee can only set this preference for the entire week and not for a single day.

Example 2.1. An example of possible combinations of preferences for an employee is given. Green represents preference to work and red represents the preference to be off.

	Monday	Tuesday	Wednesday	Thursday	Friday	Saturday	Sunday
Night							
Morning							
Day							
Evening							

Table 1: Conflicting preferences to work, green, and to be free, red.

As can be seen from the table some preferences conflict. In this case the Tuesday evening and Friday morning have the conflicting preferences to be free and to work. In this case the shift will be considered as a shift without preferences. Two other shifts that "conflict", Tuesday morning and Friday evening, both have the preference to either work twice or be free twice. This will be considered the same as only one preference. However, just like with the priority in preferences, this can be changed if deemed necessary.

2.4 Skills

Finally we are also considering skills. Both employees and shifts can be given a set of skills. A shift that requires a set of skills g_s can only be filled by employees with skill set g_e such that $g_e \supseteq g_s$, i.e, the employee has at least all the skills required for the shift. It will be preferred to find an employee with $g_e = g_s$, but if this is not possible an overqualified employee should be assigned to the shift.

Example 2.2. Let there be a shift s that requires skill g_1. If there are two possible skills, g_1 and g_2, for an employee, there are four possibilities for employees $e_1, \ldots, e_4$. Assume they have skill sets $\emptyset, \{g_1\}, \{g_2\}$ and $\{g_1, g_2\}$ respectively. In this case e_2 and e_4 are the only employees that have the required skills to work shift s. So when assigning an employee to this shift it should be either e_2 or e_4, with e_2 having priority because e_4 is overqualified.

3 Linear programming

In this chapter an introduction to linear programming and its history are given and some of the algorithms are explained. A description of the system used in the thesis is given in Section 5.2.

3.1 Introduction

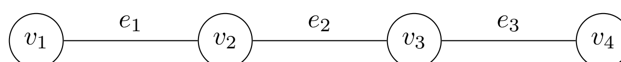
Linear programming can be used to find an optimal solution of the objective function under some constraints, both the constraints and the objective function must be linear. This can be used for minimizing the distance travelled for trucks, maximizing the profit of your company, minimizing the cost for a feasible roster etc. Also many problems in graph theory correspond to a linear programming problem, e.g. maximum matching or maximum flow.
Because minimizing and maximizing are easily interchangeable, $\max f = \min(-f)$, we will only consider maximizing. In our case it is to maximize the value of schedules to obtain an optimal roster.

Example 3.1. Your company sells two items, which we will call x_1 and x_2. You currently have 7 items of x_1 and 6 items of x_2 in stock. You profit 3 dollars from selling x_1 and 2 dollars from selling x_2. Due to limitations only 9 sales can be processed in one week. To maximize the profit, the following model can be used.

$$\begin{aligned} \max \quad & 3x_1 + 2x_2 \\ \text{s.t.} \quad & x_1 \leq 7 \\ & x_2 \leq 6 \\ & x_1 + x_2 \leq 9 \\ & x_1, x_2 \geq 0. \end{aligned}$$

The optimal solution for this problem can be solved fairly easy: sell x_1 until you don't have any left then sell x_2 until the maximum of 9 items is reached. This gives an optimal solution of 25 for $x_1 = 7$ and $x_2 = 2$.

Example 3.2. A matching in graph theory is a subset of edges such that for each vertex v at most one edge is adjacent to the vertex v.



In the graph above the possible matchings are $\{e_1\}, \{e_2\}, \{e_3\}$ and $\{e_1, e_3\}$. A maximum matching is of a matching of maximum size. In this case that is $\{e_1, e_3\}$.
To solve this with linear programming we can use the following.

$$\begin{aligned} \max \quad & \sum_{e \in E} x_e \\ \text{s.t.} \quad & \sum_{e \text{ adjacent to } v} x_e \leq 1 \qquad \forall v \in V \\ & x_e \in \{0, 1\}. \end{aligned}$$

In this example it translates to

$$\begin{aligned} \max \quad & x_{e_1} + x_{e_2} + x_{e_3} \\ \text{s.t.} \quad & x_{e_1} \leq 1 \\ & x_{e_1} + x_{e_2} \leq 1 \\ & x_{e_2} + x_{e_3} \leq 1 \\ & x_{e_3} \leq 1 \\ & x_{e_1}, x_{e_2}, x_{e_3} \in \{0, 1\}. \end{aligned}$$

The maximum is indeed found for $(1, 0, 1)$ as it is feasible and x_{e_2} can't be added because this will violate the second and third constraint.

Every system can be written in its **canonical form**. In the case of Example 3.1 this would be

$$\begin{aligned} \max \quad & \mathbf{c}^T\mathbf{x} \\ \text{s.t.} \quad & A\mathbf{x} \leq \mathbf{b} \\ & \mathbf{x} \geq 0, \end{aligned}$$

with

$$\mathbf{c}^T = \begin{pmatrix} 3 & 2 \end{pmatrix}, \mathbf{x} = \begin{pmatrix} x_1 \\ x_2 \end{pmatrix}, \mathbf{A} = \begin{pmatrix} 1 & 0 \\ 0 & 1 \\ 1 & 1 \end{pmatrix} \text{ and } \mathbf{b} = \begin{pmatrix} 7 \\ 6 \\ 9 \end{pmatrix}.$$

Given a linear program in its canonical form we can define its dual system as

$$\begin{aligned} \min \quad & \mathbf{b}^T\mathbf{y} \\ \text{s.t.} \quad & A^T\mathbf{y} \geq \mathbf{c} \\ & \mathbf{y} \geq 0. \end{aligned}$$

The reason for using the dual of a system is to find an upper bound. This will be explained by using an other example in which we create the dual of the system in Example 3.1.

Example 3.3. To start creating the dual, constraint i will be multiplied by y_i for all i. To assure the inequalities still hold we must add the constraints $y_i \geq 0$. Taking the sum of all the constraints from Example 3.1 multiplied by y_i gives the following inequality.

$$(y_1 + y_3)x_1 + (y_2 + y_3)x_2 \leq 7y_1 + 6y_2 + 9y_3.$$

To make sure this is an upper bound of $3x_1 + 2x_2$ the following inequality must hold.

$$3x_1 + 2x_2 \leq (y_1 + y_3)x_1 + (y_2 + y_3)x_2.$$

Hence we find the constraints $3 \leq y_1 + y_3$ and $2 \leq y_2 + y_3$ and this results in the dual system of

Example 3.1.

$$\begin{aligned} \min \quad & 7y_1 + 6y_2 + 9y_3 \\ \text{s.t.} \quad & y_1 + y_3 \geq 3 \\ & y_2 + y_3 \geq 2 \\ & y_1, y_2, y_3 \geq 0. \end{aligned}$$

This leads us to the following two duality theorems. The first can be proven by generalizing the example and the second with the use of Farkas' Lemma, see [3].

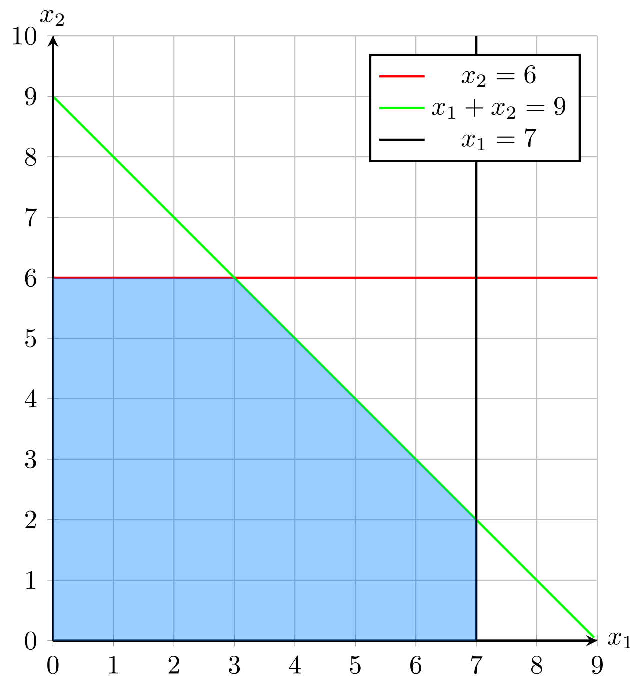
Theorem 3.4. Weak duality theorem
Given a feasible solution $\mathbf{x}$ for the primal system and a feasible solution $\mathbf{y}$ for its dual system we have $\mathbf{c}^T\mathbf{x} \leq \mathbf{b}^T\mathbf{y}$ i.e.

$$\max \mathbf{c}^T\mathbf{x} \leq \min \mathbf{b}^T\mathbf{y}.$$

Theorem 3.5. Strong duality theorem
If the primal or dual problem has a finite optimal solution then so does the other and then

$$\max \mathbf{c}^T\mathbf{x} = \min \mathbf{b}^T\mathbf{y}.$$

An optimal solution will always lie in the **feasible region**, which is the set of all solutions: $\mathcal{F} = \{\mathbf{x} | A\mathbf{x} \leq \mathbf{b}, \mathbf{x} \geq 0\}$. Example 3.1 has the following feasible region.



Many algorithms use the feasible region to find an initial solution and improve that initial solution to eventually obtain an optimal solution.

3.1.1 Integer linear programming

So far we have seen examples which had an integral optimal solution. In general the solutions of an LP don't have to be integral as we will see in Example 3.6.

Example 3.6.

$$\begin{aligned} \max \quad & x_1 + x_2 \\ \text{s.t.} \quad & x_2 \leq 1 \\ & 2x_1 + x_2 \leq 2 \\ & x_1, x_2 \geq 0. \end{aligned}$$

The point $(\frac{1}{2}, 1)$ lies in the feasible region and gives an value of $\frac{3}{2}$ which is higher than all the integral solutions $(0,0), (1,0)$ and $(0,1)$ which have values 0,1 and 1 respectively. In fact, one can show that $\frac{3}{2}$ is the optimal solution.

In a lot of applications of linear programming an integral solution is required. You can't partially sell an item or partially assign a shift to an employee.
This is one of the reasons to make a distinction between linear programming, integer linear programming, mixed integer programming and binary programming (or 0-1 integer programming).
In linear programming there are no restrictions in the solutions; integer programming requires the solution to be integral, whereas mixed integer programming only requires a part of the solution to be integral. Binary programming is a special case of integer programming where all variables must be binary.
An other reason for this distinction is the fact that algorithms that can be used for linear programming in general can't always be applied to integer programming.
As we will see in Section 3.2 an optimal solution for a linear programming problem can be found in polynomial time. However, integer linear programming is proven to be NP-complete, as 0-1 integer programming is one of Karp's NP-complete problems.[4] There are several algorithms that are used to solve ILPs. One commonly used method, branch-and-bound, is described in Section 3.3.

3.2 History

A way to solve a system of linear inequalities was first introduced in 1827 by Joseph Fourier (1768–1830), who used the rewriting of inequalities to eliminate the variables one by one. This is known as the Fourier-Motzkin elimination, named after Fourier and Theodore Motzkin (1908–1970), who also introduced this method of elimination independently in 1936. More details about the Fourier-Motzkin elimination can be found in Section 3.3.1.
Fourier wasn't the only mathematician that considered linear inequalities. Lagrange's multiplier theorem inspired multiple mathematicians to consider the case for inequality constraints. Examples are Cournot and Ostrogradsky, who also made attempts to prove Farkas' lemma, which eventually was proven by the Hungarian Gyula Farkas (1847–1930). This lemma can be used to prove the strong duality theorem, which in its turn can be used to find an optimal solution faster using the dual of a system.[5]
Although some progress concerning linear inequalities was made, the first formulation of a linear program used today wasn't described until 1939. This year Leonid Kantorovich (1912–1986) formulated the problem that is equivalent to the linear programming problems that we still use. Therefore he is sometimes regarded as the founder of linear programming.
During World War II linear programming made great developments due to the necessity of optimal

planning in the military. But it wasn't until after World War II that really big steps were made. George Dantzig (1914–2005) is mostly known for his development of the simplex algorithm and seen as the father of linear programming.[6] After finishing his Ph.D. he was looking for a job and the idea came up to mechanize the planning process, which until then was all done by hand.
In 1947 Dantzig proposed the simplex method. A method that, although it doesn't have a polynomial running time in the worst case, still proves to be one of the most efficient methods to find a global optimum. A discription of the simplex method can be found in Section 3.3.2.
An example of what linear programming was used for at the time, is the Berlin Airlift. This was the problem of being able to supply Berlin during the time this was only possible by plane.
Also in 1947, Dantzig met John von Neumann (1903–1957) to talk about linear programming. Von Neumann immediately recognized parallels between game theory and linear programming. For this reason he also thought of the theory of duality in linear programming. He also suggested another method to solve a system. This was the first interior-point method, but didn't run in polynomial time. It even turned out to be much less efficient than the simplex method of Dantzig.
An interior-point method found by Narendra Karmarkar (1955–) in 1984, known as Karmarkar's algorithm, was the first polynomial time algorithm that proved to be useful in practice. However this wasn't the first polynomial time algorithm. The first algorithm that runs in polynomial time is the ellipsoid method and was first described by Leonid Khachiyan (1952–2005) in 1979.
Today linear programming is used in many places to maximize profit and minimize time and for many other applications. But still the research is far from finished and new applications are made and old ones are being improved.

3.3 Algorithms

In this section some algorithms that were important breakthroughs and some of which are still being used are discussed in this section.

3.3.1 Fourier-Motzkin

This method, first suggested by Fourier, eliminates variables one by one. This algorithm will be explained per example. Note that we are not looking for an optimal solution, but to reduce the number of variables.

Example 3.7. Assume that there are four constraints:

$$\begin{aligned} x_1 + 2x_2 - 3x_3 &\leq 1 \\ -x_1 + 3x_2 + \ x_3 &\leq 2 \\ 2x_1 - \ x_2 - 2x_3 &\leq 3 \\ -2x_1 + \ x_2 + \ x_3 &\leq 4. \end{aligned}$$

First we will eliminate x_1. To do this we will rewrite the inequalities to

$$\begin{aligned} x_1 &\leq 1 - 2x_2 + 3x_3 \\ x_1 &\geq -2 + 3x_2 + \ x_3 \\ x_1 &\leq (3 + \ x_2 + 2x_3)/2 \\ x_1 &\geq (-4 + \ x_2 + \ x_3)/2. \end{aligned}$$

This results in the following combinations of inequalities.

$$\begin{aligned} -2 + 3x_2 + \ x_3 &\leq 1 - 2x_2 + 3x_3 \\ (-4 + \ x_2 + \ x_3)/2 &\leq 1 - 2x_2 + 3x_3 \\ -2 + 3x_2 + \ x_3 &\leq (3 + \ x_2 + 2x_3)/2 \\ (-4 + \ x_2 + \ x_3)/2 &\leq (3 + \ x_2 + 2x_3)/2. \end{aligned}$$

This results in a new system with one fewer variable.

By combining the linear inequalities that are expressed in terms of one variable, x_1 in the example, the number of inequalities increases in general.
Assume that there are n inequalities and, when written in terms of x_1, they are spread evenly concerning $\leq$ and $\geq$. There are then $(\frac{n}{2})^2$ new inequalities found. Repeating this step for k variables a worst case of $\mathcal{O}((\frac{n}{2})^{2^k})$ inequalities are created.

3.3.2 Simplex

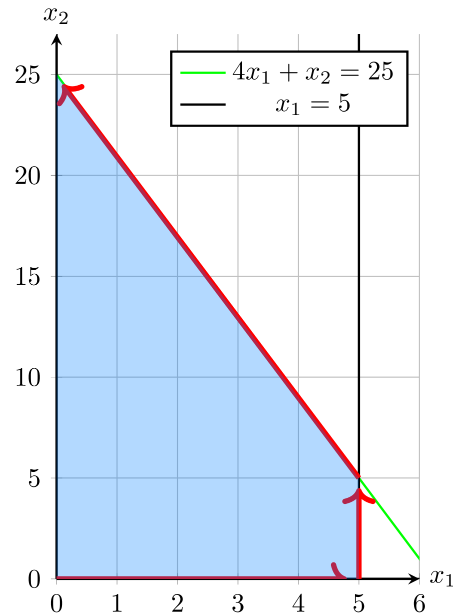
To explain the simplex method in detail a lot of terminology must be introduced. For this reason a rough sketch and intuitive explanation will be given and for a very detailed explanation we refer to A. Schrijver[5].
Before sketching the simplex method it is important to realize that an optimal solution can always be found in a vertex of the feasible region. Because if you have a solution and are not on a vertex there is a variable for which increasing and decreasing its value is possible. Travelling one of these directions will not decrease the value of the solution because the system is linear. Repeating this process will finally result in a solution in a vertex.
A vertex is therefore called a basic feasible solution(BFS).

Once a BFS is selected the idea is to travel from this vertex to a neighbouring vertex. Which neighbour is chosen during this so called pivot operation can depend on different factors. Of course the first option is to check if an increase in the value of the solution is possible. However if this isn't possible this may result in stalling, not increasing the value, or cycling, repeating the same pivot operations. To avoid cycling and assure termination of the algorithm, Bland's rule can be used.[5]

The complexity of the simplex method however remains exponential in the number of variables. This was illustrated by Victor Klee and George Minty with the Klee-Minty cube in 1972.[5] This deformed cube in n dimensions is defined by

$$\begin{aligned} \max \quad & 2^{n-1}x_1 + 2^{n-2}x_2 + \dots 2x_{n-1} + x_n \\ s.t.\ & x_1 \leq 5 \\ & 4x_1 + x_2 \leq 5^2 \\ & \vdots \\ & 2^n x_1 + \dots + 4x_{n-1} + x_n \leq 5^n. \end{aligned}$$

Starting at $\mathbf{0}$ all of the 2^n vertices will be travelled before the optimal solution of 5^n is found in vertex $(0, \dots, 5^n)$. The cube and path travelled for $n = 2$ are given. Note that the first step moves to (5,0), as increasing x_1 increases the solution twice as much as increasing x_2 would.



3.3.3 Ellipsoid method

The ellipsoid method was proven to be running in polynomial time by Khachiyan. Although it turns out that the method can't be relied on in practice, as small changes in values, due to rounding, have a large impact that may result in finding the wrong solution. The simplex method is in general still faster than this method even though we just saw that the simplex method has exponential running time in the worst case.
Because of its importance as the first algorithm to be proven to run in polynomial time, it should

be mentioned, but because of it not being practical it will be explained only briefly.
The method starts with an ellipsoid that contains an optimal solution. As a starting point an ellipsoid containing the entire feasible region can be used. With each step a new ellipsoid is selected such that it contains half of the previous ellipsoid and the optimal solution. The maximum number of iterations depends on the distance allowed from the optimal solution and the size of the initial ellipsoid, but is in any case polynomially bounded.[7]

3.3.4 Branch-and-bound

Note that there is not one branch-and-bound algorithm, but multiple algorithms that use the same steps. Branch-and-bound algorithms are used to find an optimal integer solution for a system. As we've seen in Example 3.6 an optimal solution found with, for example, the simplex method, isn't always integral. The branch-and-bound algorithms consist of several different steps.
The first is to branch. This means that a feasible region is split in multiple regions. Preferably disjoint, to prevent considering the same point multiple times, but this is not required.
An other possible step is to bound. At this step an optimal solution is found for one of the branches. Note that this solution isn't necessarily integral.
The last possibility is to find an integral solution for a region.
Bounding and finding an integral solution are used to prune the tree when required. This will be explained by looking at the system from Example 3.1.

Example 3.8. We are looking at

$$\begin{aligned} \max \quad & 3x_1 + 2x_2 \\ \text{s.t.} \quad & x_1 \leq 7 \\ & x_2 \leq 6 \\ & x_1 + x_2 \leq 9 \\ & x_1, x_2 \geq 0. \end{aligned}$$

As a first step we will branch, creating two new problems. The first system S_1 will have the added constraint $x \leq 3$. The second, S_2, has constraint $x > 3$, or $x \geq 4$ as the solution must be integral.
As a next step we can bound region S_1. We claim that the optimal solution is 21 at (3,6).
An other step is finding an integral solution, we will do this for S_2. A possible, but not necessarily optimal, solution is 22 in (6,2).
Now that there is an integral solution in S_2 that is greater than the optimal solution in S_1 we can conclude that studying S_1 any further is useless. For this reason we will prune this branch and continue branching/bounding and finding solutions until an optimal integral solution is found.

As stated at the beginning of this section there is not one branch-and-bound algorithm. Determining when to branch, bound or find an integral solution and on which subsystem this should be done depends on the shape of the linear programming system. For this reason many different variations have been developed.

4 Creating possible schedules

The first step to finding a roster, for all employees combined, is creating the possible schedules, for an individual employee. The problem of creating all possible schedules for the employees is exponential in the number of possible shifts. If we would brute force this without any restrictions, we would find $n \cdot 2^{s'}$ schedules, where s' is the number of different shifts during the week and n the number of employees. This holds because adding a shift is independent of other shifts for each employee. However, obviously we don't need all of these schedules. They could deviate too much from the contract hours of an employee. If the number of different shifts becomes larger there will be more overlaps in the shifts as well. And a schedule is only feasible if it doesn't violate any of the working regulations.
In this chapter it will be explained how these issues are tackled when creating the possible schedules.

4.1 Category of schedules

First of all, to reduce the number of schedules to be created, **categories of schedules** are used. Categories partition the employees based on their characteristics. This is inspired mainly by nurse rostering problems [8]. Instead of creating all schedules for every employee we will be creating all schedules for every category. Because all the employees in the same category are essentially the same they will also have the same possible schedules.

The categories created will depend on: contract hours, days off, holidays, preferences and whether they work night shifts or not.
Recall that the possible preferences are to (not) work on a (part of a) day. All of those four combinations are considered when creating the categories.

Using categories creates the following problem: the **value of a schedule** should depend on both the schedule and the employees kudos. To solve this, the value v_s will for now solely depend on the schedule s and not on the employee e. This will be determined depending on preferences, but can be altered to depend on other factors too. How v_s is determined, is explained in Section 5.1.1. The value v_s and the amount of kudos k_e will then be used to get a personalized value $c_{s,e}$.

One might notice that the skills of an employee are not considered when creating the categories. This will be done inside the linear programming model. There are a few advantages to postponing this distinction until then. First of all, if the category doesn't depend on the skills more people will fall in the same category, thus decreasing the number of different categories, and therefore schedules, that must be made. An other advantage is that a lot of similar combinations won't be created as shown the following example.

Example 4.1. Let e be an employee with a single skill g and let $s_1, \ldots, s_{2k}$ be $2k$ shifts. Let shifts s_{2i-1} and s_{2i}, $1 \leq i \leq k$, have the same start and end time, but s_{2i-1} requiring no skills and s_{2i} requiring one skill g.
If we don't consider the skills then s_{2i-1} and s_{2i} are equal and thus there are at most 2^k possible schedules. However if we do consider skills there is an upper bound of 2^{2k}.
When there are n skills, there are 2^n possible subsets of skills. So when concerning skills while making possible schedules increases the upper bound of 2^k to $2^{2^n k}$ which isn't doable even for a very small k and n.

4.2 Overview model

To create all the possible schedules the following model is used. Below an overview of the objects is displayed. However, one should note that this is a simplified version. In the working model there are some more possibilities, like having multiple scenarios that can be used interchangeably. However because this is not necessary in this thesis, the model has been simplified.

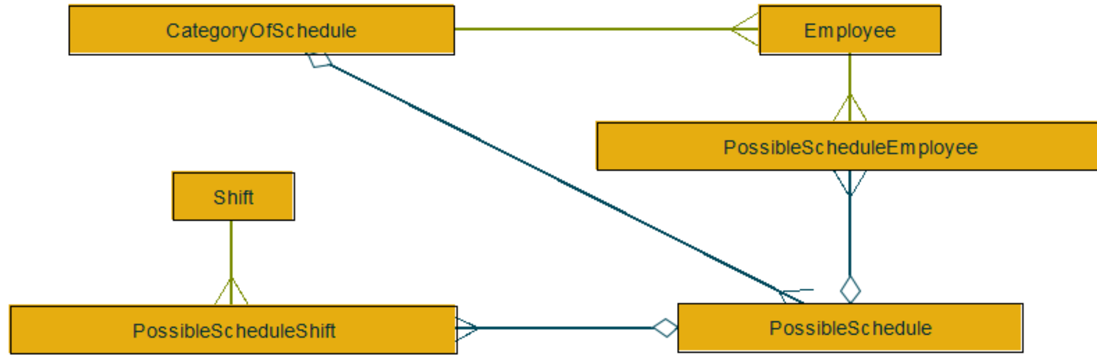


Figure 1: Objects used in the model

To explain the full structure we start at `CategoryOfSchedule`. There is a connection from `Category-OfSchedule` to `Employee` by a so called 1-N relation. Each `Employee` is connected to one `CategoryOf-Schedule` and every `CategoryOfSchedule` can be connected to multiple `Employees`.
Each `CategoryOfSchedule` can have multiple `PossibleSchedules`. So in the same way there is a 1-N relation between `CategoryOfSchedule` and `PossibleSchedule`. Note that the path from `Employee` to `CategoryOfSchedule` is unique. Thus all the `PossibleSchedules` for an `Employee` can be found via `CategoryOfSchedule`.
For all `PossibleSchedules` and `Employees` that are related the value $c_{s,e}$ must be saved. This is done by using the object `PossibleScheduleEmployee`. This object helps to create an N-M relation between `PossibleSchedule` and `Employee`. It can be seen as if one `PossibleScheduleEmployee` is a pair (s, e) that is connected to the corresponding `PossibleSchedule` s on one hand and the `Employee` e on the other.
A `PossibleSchedule` consists of several `Shifts` and of course a `Shift` will be placed in multiple `PossibleSchedules`. For the same reason as before the object `PossibleScheduleShift` is used to create this N-M relation.
Finally there is a slight difference between the blue and green relations. The blue relations are owning relations. This means, e.g, that `CatergoryOfSchedule` owns `PossibleSchedule`. The reason behind this is that every object must be owned by an other to save its data. The objects `CategoryOfSchedule`, `Employee` and `Shift` are owned by other objects that are not relevant for now.

4.3 Possible schedules

Recall that the number of possible schedules is exponential in the number of shifts. To increase the speed for finding all the possible schedules the working regulations are used. This is in general a very limiting factor to the working schedules for employees. The minimum breaks between shifts rule

out most of the possibilities and even decreases an exponential number of schedules to a polynomial number of schedules and an algorithm that runs in polynomial time as seen later in this section.

4.3.1 Reducing the number of shifts

As the time complexity depends on the number of shifts, a reduction in the number of shifts is important. For this reason the unique shifts will be used to create the possible schedules. This means that if a shift is required multiple times, i.e, multiple employees are required to work at the same time, this shift will be used once. All possible schedules will still be found and no duplicate possible schedules will be created.

Just as with the categories, the required skills for a shift will not be considered yet. For the same reasoning as above this will increase speed and create no duplicates, although it is less trivial that this doesn't cause a problem and will be solved correctly. This way an employee without skills could get a possible schedule with a shift that does require a skill. The reason that this will not become a problem is illustrated in Example 5.6.

4.3.2 Algorithm

Now that the set of shifts only differ in either start or end time of the shift the following algorithm will be run.

Algorithm 1: CreateSchedules

```
Result: All possible schedules for a category C.
S := shifts ordered by starttime;
PosSched := ∅;
RecursiveCreateSchedules(0, S, PosSched);
```

Algorithm 2: RecursiveCreateSchedules(i, S, PosSched)

```
Result: If not all shifts have been considered: continue without shift i and with shift i in a
        potential schedule. Otherwise: save schedule if it is feasible.
if i < #S then
    RecursiveCreateSchedules(i+1, S, PosSched);
    PosSched:= PosSched ∪ S[i];
    i := Increment(i, S);
    RecursiveCreateSchedules(i, S, PosSched);
else
    if ContractHasBeenMet(PosSched) then
        Add(PosSched);
    end
end
```

The function `Increment` increases i to the first shift that isn't on the same day and also satisfies the 9 hour break that is required.
Because of the first step of `Increment` the function `CreateSchedules` will run in polynomial time in the number of shifts. Without incrementing i the algorithm would run in $\mathcal{O}(2^{\#S})$ time. Now that each day has at most one shift, the function `RecursiveCreateSchedules` will be called $\mathcal{O}(s^7)$ times,

where s denotes the maximum number of shifts on a day. The computation speed of `Increment` is at most $\mathcal{O}(\#S) = \mathcal{O}(s)$. `ContractHasBeenMet` is a check if the schedule contains a weekend and the right number of contract hours. This is done in constant time. `Add` and $\cup$ are also done in constant time.
Hence the algorithm is of $\mathcal{O}(s^8)$.

To reduce the time complexity of the algorithm even more two methods are considered.
The first idea is to split the week in two parts and combine those later on. Splitting a week in 4 days and 3 days the algorithm would be $\mathcal{O}(s^5)$ and $\mathcal{O}(s^4)$ respectively. As the number of schedules found has an upper bound of s^4 and s^3 the time required to combine these is $\mathcal{O}(s^4 \cdot s^3) = \mathcal{O}(s^7)$.
Note that the check if a schedule is indeed correct is done in the end where the schedules are combined. As a weekend might occur on the 4th and 5th day this can't be done earlier.
That's where the second idea comes in. To overcome the problem of many schedules being rejected, a weekend can be enforced. This will be done by creating six subcategories each having a weekend on two different days. The way this is done is by selecting days off on the days the weekend should occur.
Note that there are only six possible ways to have a weekend because there is no data about earlier weeks.

Algorithm 3: CreateSchedulesFromSubCategories

```
Result: All possible schedules for a category C
SubC := Subcategories of C;
for c in SubC do
 |  CreateSchedules(c);
end
Merge(SubC);
RemoveDuplicates(C);
```

Observe that for all of these three methods the same possible schedules are created. Now for the time complexity. First of all we should note that, because we lost two days, `CreateSchedules` now runs in $\mathcal{O}(s^6)$. For each of those categories there are at most s^5 possible schedules. So `Merge` will be done in at most $\mathcal{O}(s^5)$. The bad news is that `RemoveDuplicates` will be of order $\mathcal{O}(s^{10})$.
The way `RemoveDuplicates` works is explained in more detail in Section 4.3.4.

In this scope `RemoveDuplicates` is not always required. Recall that someone working full time has 5 shifts spread out over different days. So there is no way two possible schedules that are the same occur in two different subcategories, and no double schedules will be created within one category. So in this case there is no reason to check for duplicates.
Part time employees can have at most 3 shifts. So in this case removing duplicates is of order $\mathcal{O}(s^6)$. Because of these facts the method using subcategories is used to find possible schedules.

4.3.3 Holidays

As briefly mentioned earlier we will consider holidays as days off but treating the contract hours differently. Some small examples will give an idea of the possibilities.

Example 4.2. Assume that a full time employee has a holiday on Tuesday.
A possible schedule could contain shifts on Monday and Wednesday through Friday. This comes from a schedule having shifts from Monday through Friday and removing the holiday.

An other possible schedule could be to work Wednesday through Sunday as it is allowed to let the holiday be part of the weekend.
However a schedule from Wednesday through Friday is not allowed as there is no possible schedule that can be reduced to this schedule by removing the shifts on Tuesday.

From the example above can be seen that we can't simply pick all the schedules that have fewer than the required contract hours. So as a first step the schedules without holidays will be created and then the holidays will be removed.

Algorithm 4: CreateSchedulesHolidays

Result: All possible schedules for a category C with holidays
C' := C without holidays;
CreateSchedules(C');
CopyAndRemoveShifts(C,C');
RemoveDuplicates(C);

Copying schedules from category C' to C and removing shifts is linear in the number of schedules and thus doesn't increase the time complexity we found earlier.
However one important note is that we can't ignore removing duplicates for full time employees as done when they didn't have holidays set. The following example will explain why.

Example 4.3. Some possible schedules are given:

	Monday	Tuesday	Wednesday	Thursday	Friday	Saturday	Sunday
0:00–8:00							
6:00–14:00							
12:00–20:00							
18:00–2:00							

Table 2: Four possible schedules in their own color.

Considering weekends and the time between shifts it's clear that each schedule is indeed a possible schedule. A holiday on Thursday would make blue and yellow the same schedule.
If there are more holidays in one week even days can be interchanged as illustrated by the green and red shifts where holidays on Friday and Sunday results in the same schedule.

So if you don't want any duplicate schedules they need to be removed for full time employees as well.

4.3.4 Removing duplicates

When looking for duplicates every pair of schedules can be checked in $\mathcal{O}(n^2)$ time. Where n is the number of schedules. Although not reducing the time complexity, the speed can be increased using some heuristics.
Every possible schedule s has some properties that can be used to conclude two schedules must be different. The properties used are:

- number of shifts;
- start of the first weekend;
- start first shift;
- start last shift;
- the value v_s.

A difference in one of these points implies two different schedules. Ordering the schedules based on these properties takes $\mathcal{O}(n \log(n))$ time. They can then be separated in different groups and are checked separately.

Example 4.4. To show how the schedules for a category are made we use this example. Given a category C with the properties that there is a day off on Wednesday and a holiday on Thursday the following will happen.
First off, a category C' will be created with the property of a day off on Wednesday.
To create the schedules for this category the subcategories C_1 through C_6 will be created. Concerning weekends and days off these will have the following days off.

	Monday	Tuesday	Wednesday	Thursday	Friday	Saturday	Sunday
C_1	■	■	■				
C_2		■	■				
C_3			■	■			
C_4			■	■	■		
C_5			■		■	■	
C_6			■			■	■

The possible schedules for these categories are then created. Note that for a full time employee this results in 0 possible schedules for every category except C_2 and C_3 because they are required to work 5 shifts. The schedules are copied to C' and, if necessary, duplicates are removed.
Finally the possible schedules from C' will be copied, get their shifts on Thursdays removed and added to C. After removing duplicates all the possible schedules for C haven been created.

4.3.5 Variations

There are some variables that can be edited that don't change complexity of the algorithm and can be adjusted depending on the number of schedules wanted.
One of these in the **maximum violations** in the possible schedules. Preferences haven't been mentioned while creating possible schedules and so far are only used to calculate the value of the schedule. However it is possible to set a maximum number of violations of preferences. Whenever

the maximum has been violated the schedule won't be saved. The advantages of this are that we will find fewer schedules, find them faster and the schedules that are found are the best for the category. This should however be handled with care. Sometimes an employee might have to work more shifts he doesn't prefer than allowed to satisfy the roster. In some extreme cases it might not be possible to find any schedules at all as seen from the following example.

Example 4.5. Let's assume we set the maximum violations to 3 and a full time employee prefers to be free during the evening. However, if the only shifts that week are in the evening, every schedule created for this employee will violate the preferences 5 times, so no possible schedules will be created.

A second variable of freedom is the **deviation from the contract hours** h. If we set this to t hours we allow every possible schedule that has a duration in the interval $[h - t, h + t]$. Recall that in this thesis $t = 5$.
Because the duration of a schedule is checked in the end, it doesn't change the speed of the algorithm. But it will change the number of schedules found. This is again a trade off between having fewer but better schedules and more schedules, which implies more freedom.

5 Finding a roster

5.1 Schedule value and kudos

It is now time to find the fairest roster possible. Which immediately brings us to a hard question: what is fair?
Of course there are some things that most people will agree on. E.g, it wouldn't be fair if one employee gets his preferences every week and another never does. So that's something that should be avoided.
But on other subjects the fairest solution isn't that obvious and probably the answer will depend on whom you ask. For example concerning the number of preferences a person has. If an employee has only one preference in the week and an other has, let's say, five preferences. Should both these preferences treated equally because all employees are equal? Should the employee with one preference be slightly prioritized over the second employee because losing one of five preferences isn't as bad as losing the only preference?
Another conflict is the number of contract hours compared to preferences. Should not getting your preference be considered worse for a part timer than for a full timer?
Should a second bad shift be considered just as bad as the first? One could argue that it decreases the value of a schedule more than the first bad shift.
How do we treat employees that don't have any preferences? Shouldn't we change their amount of kudos at all? Perhaps they deserve a small reward because creating a roster is now easier?
There are also other factors that don't depend on preferences, such as the value of night shifts, long weekends, or the hours worked compared to the contract hours. The list goes on and on and not all the options can be considered. For this reason only the preferences will be considered, and looking at differences such as part time compared to full time and the number of not preferred shifts.

Using the value of a schedule and previous rosters determines which employee should get a better schedule compared to other employees. This will be done in three steps. First of all we create a value $c_{s,e}$ which depends on the schedule and the employee. Then, using these $c_{s,e}$, we find the fairest roster. Finally we check how good the given schedules are compared to other employees' schedules and this will be monitored using kudos.

5.1.1 Calculating the value of a schedule

The first thing to do while creating an honest distribution is distinguishing the number of preferences. To do this, we set the value of a preferred shift to 2, a shift that is not preferred to 0 and set the value to 1 if no preference was given.

Example 5.1. Let there be an employee that prefers to work in the evening, so a start time between 13.00 and 20:59. And who prefers not to work on a Monday. The following table shows some possible schedules with corresponding values of shifts.

	Monday	Tuesday	Wednesday	Thursday	Friday	Saturday	Sunday
0:00–8:00							1
6:00–14:00	0	1	1				1
12:00–20:00				1	1	1	1
18:00–2:00	1	2	2	2	2	2	

Table 3: Examples of schedules with their value.

Recall that a shift on Monday evening will be valued as a shift without preferences. Taking the sum of the values of the shifts we get $v_{\text{green}} = 8, v_{\text{yellow}} = 5, v_{\text{orange}} = 5$ and $v_{\text{red}} = 9$.

This method results in a value $v_s \in [0, 2s']$, where s' is the number of shifts of the schedule s.
Note that at this point working one bad shift for a full time or a part time employee has the same cost. Also losing a preferred shift, or gaining a preferred one, is the same for every shift, independent of the number of not preferred shifts. The distinction between these possibilities occurs when determining the $c_{s,e}$.
Something else worth noting is that getting one preferred shift and one bad shift currently has the same value as no preferences, as can be seen from v_{yellow} and v_{orange} in the example. This is something that will be considered as the same throughout the thesis, however worth considering to change in the future.

5.1.2 Kudos

Personalizing the value of a schedule is important as some employees will have priority over other employees based on previous rosters. To do this we will give every employee e kudos $k_e \in [0, 100]$ with the condition that the average amount of kudos is constant, and thus so is the total, $\sum_e k_e$, if no employees are added. The reason behind this is that it is, for one, clearer to see for an employee if the amount of kudos they have is high or low. Another reason to do this has to do with new employees. If a new employee is added they can be given the average amount of kudos, which in this case is constant, and there is no need to determine a fair amount of kudos to start with every time. We want an employee with more kudos to have priority over an employee with less kudos. This is how previous rosters can be considered. Once a new roster has been made the kudos will be updated and because $\sum_e k_e$ is constant the employees who got a relatively bad schedule must get an increase in kudos and employees with a relatively good schedule will see the amount of kudos decrease. The notion relative is very important as the same schedule may result in both an increase and decrease of the amount of kudos, as seen in the following example.

Example 5.2. Let e_1, e_2 be full time employees both with 50 kudos. Employee e_1 got assigned a schedule s_1 in which he prefers all shifts, so with $v_{s_1} = 10$. Employee e_2 got a schedule s_2 with value $v_{s_2} = 9$. So value s_2 is lower and we expect an increase in the amount of kudos of e_2.

However if at a later point in time both employees have 50 kudos, e_2 gets assigned s_2 and employee e_1 a schedule s_3 with $v_{s_3} = 8$ you expect an increase in the kudos of employee e_1 and hence a decrease in the kudos of e_2.

5.1.3 Personalised value

Now that we have both v_s and k_e the value $c_{s,e}$ can be determined. When we use this value in our linear program we want to maximize the sum of the values of the chosen schedules. This means the if we create a mapping $c : [0, 2s'] \times [0, 100] \to \mathbb{R}$ such that $c(v_s, k_e) = c_{s,e}$. We want c to be increasing in both directions, that is:

- $c(v, k) \leq c(v, k')$ whenever $k < k'$ and
- $c(v, k) \leq c(v', k)$ whenever $v < v'$.

Note that even though the value $c_{s,e}$ are used in an linear program, c itself need not be linear as $c_{s,e}$ is calculated beforehand.
The reason to let c be increasing is very intuitive. If the amount of kudos is increased the personal value should never decrease. Same goes for the value of a schedule.
For the same reason we might consider using a strictly increasing c in both directions.
There are uncountable many functions c that are (strictly) increasing. We will pick five for some different reasons mentioned below to have some insight in the different priorities one may have.
These are the functions used:

1. $c_{s,e} = v_s + k_e$;
2. $c_{s,e} = v_s \cdot k_e$;
3. $c_{s,e} = \frac{v_s}{2s'} \cdot k_e$;
4. $c_{s,e} = (\frac{v_s}{2s'})^2 \cdot k_e$;
5. $c_{s,e} = \sqrt{\frac{v_s}{2s'}} \cdot k_e$.

It's clear that all of the above functions are strictly increasing in both directions if the number of shifts remains constant. The choice of these five functions has several uses.
First of all we test the differences between addition and multiplication. Second, as mentioned before, we want to scale our value depending on part time and full time employees. This can be done by comparing the second and third method.
The third, fourth and fifth are all a different way of valuing the difference in cost of losing multiple preferred shifts in one schedule. The linear method treats all shifts equally. The quadratic method makes the impact of losing the first preferred shift higher compared to the other shifts. The square root does the opposite: getting one bad shift shouldn't be too bad, but losing more shifts will be increasingly worse.

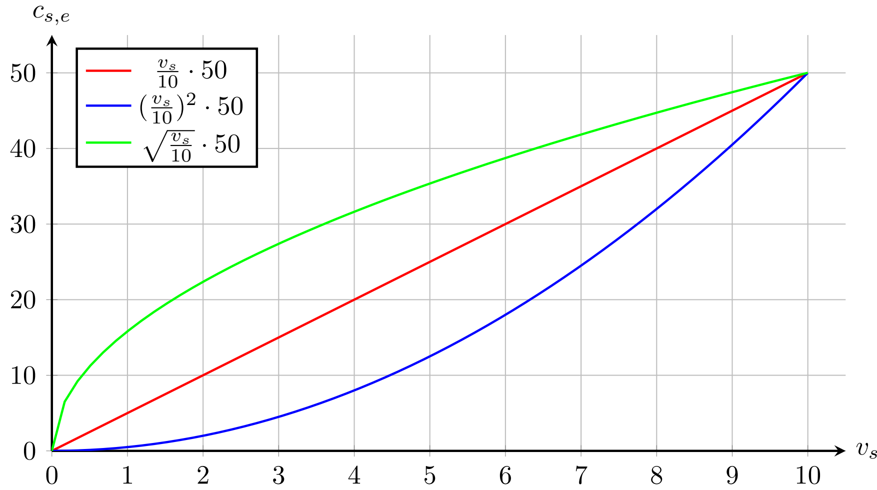


Figure 2: Values for a full time employee with 50 kudos.

One may notice that all the c are linear in k_e. The reason to keep it this way is the fact that for the same values of schedules the kudos can be recalculated in different ways. That way a variance in kudos can be created and it was decided to keep this option in one place. However this is something that can be researched later.

5.1.4 Updating kudos

For now we assume that the best roster has been found and exactly one schedule s is selected for each employee e.

To determine a way to update the kudos we should first discuss what should happen over time. For example: there are multiple employees with the same number of contract hours that have the same preferences that can't all be fulfilled. In that case it would make sense that every employee will get their preferences equally often. On top of that the amount of kudos should again be the same for all of them.
A second point is the employees without preferences. These should not be disadvantaged.
As a third we consider it fair that a part time employee gets his preference more often compared to a full time employee. Because having only a few shifts with one bad shift is worse than having more shifts with only on shift that isn't preferred.
Finally it can be the case that someone does get his preferences and an other didn't, but their preferences were independent of each other. In this case we will consider it fair that the person who got his preferences has a decrease in the amount of kudos. A more detailed explanation of this can be found in Section 6.3.2.
Recall that the total amount of kudos must remain constant. To achieve this we will first recalculate the kudos so that employees with better schedules will get less kudos added than those with worse schedules. After that the kudos will be scaled to obtain the constant amount of kudos as before to get this relative increase and decrease in kudos. Examples are given in 5.3 and 5.4.

Both of these steps can of course be done in a lot of different ways. However there are two distinctions for both cases: additive and multiplicative. To look at the different behaviour we will test four ways

to recalculate the kudos and two ways to scale them.
For **recalculating** the kudos we will use one of the following ways:

1. $k'_{n+1,e} = k_{n,e} + s' - v_s$
2. $k'_{n+1,e} = k_{n,e} \cdot \frac{s'}{v_s}$
3. $k'_{n+1,e} = \begin{cases} k_{n,e} \cdot \frac{s'}{v_s} & v_s \geq s' \\ k_{n,e} \cdot (-\frac{v_s}{s'} + 2) & v_s < s' \end{cases}$
4. $k'_{n+1,e} = k_{n,e} \cdot \frac{m_s}{v_s}$.

In the above equations we denote $k_{n,e}$ the kudos of employee e in week n. For the number of shifts in a schedule s we still use s'. Note that v_s can be equal to zero. However in reality this doesn't happen often and for the sake of clarity we write v_s instead of $\max\{1, v_s\}$ when we divide by v_s. Considering $k_{n,e}$ and s' as constants we can look at the influence of v_s on $k'_{n+1,e}$. For the first method there will be an example. The second and third have their factor by which is multiplied plotted in a graph. The last one introduces m_s which is the maximum possible value for a schedule with the same number of shifts. A more detailed explanation on this method is discussed in Section 6.3.1.

Example 5.3. The full time employees e_1, e_2 got assigned schedules s_1, s_2 respectively and $k_{n,e_1} = k_{n,e_2} = 50$ and $v_{s_1} = 7$ and $v_{s_2} = 10$.
Using the first way to recalculate the kudos will result in $k'_{n+1,e_1} = 50 + 5 - 7 = 48$ and $k'_{n+1,e_2} = 50 + 5 - 10 = 45$.
To be continued in Example 5.4.

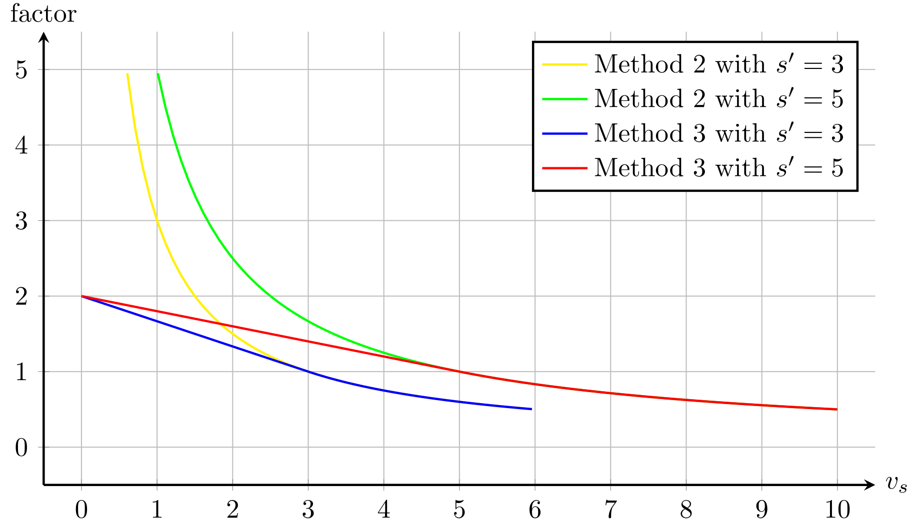


Figure 3: Factor of multiplication of kudos depending on the value of the schedule.

The second and third method will result in a increase of $k'_{n+1,e}$ compared to $k_{n,e}$ whenever $v_s < s'$ and a decrease whenever $v_s > s'$.

The third method was added to determine if there are any differences between different bounds of c. For this reason the third method keeps the factor, by which is multiplied, fixed between $\frac{1}{2}$ and 2.

The second step, **scaling** the kudos, is fairly straight forward. This is used to keep the amount of kudos equal from week to week. As mentioned before we test this using addition and multiplication:

1. $k_{n+1,e} = k'_{n+1,e} + \frac{\sum_e (k_{\text{average}} - k'_{n+1,e})}{\#e}$
2. $k_{n+1,e} = k'_{n+1,e} \cdot \frac{\sum_e k_{\text{average}}}{\sum_e k'_{n+1,e}}$.

This does not guarantee that $k_{n+1,e} \in [0, 100]$. To solve this finally let

$$k_{n+1,e} = \max\{0, \min\{100, k_{n+1,e}\}\}.$$

Of course this could result in a loss of the average wanted. In reality this doesn't occur often and if so the kudos are at such a height (or low) that a good (or bad) schedule compensates for this immediately and it is fixed the next week. However for this reason it is important to scale using $\sum_e k_{\text{average}}$ and not $\sum_e k_{n,e}$.

Example 5.4. Continuing from Example 5.3. Assume $k_{\text{average}} = 50$.
The additive scaling results in

$$k_{2,e_1} = 48 + \frac{2+5}{2} = 51.5, \qquad k_{2,e_2} = 45 + \frac{2+5}{2} = 48.5.$$

Multiplicative scaling results in

$$k_{2,e_1} = 48 \cdot \frac{100}{93} \approx 51.61, \qquad k_{2,e_2} = 45 + \frac{100}{98} \approx 48.39.$$

In both cases the average kudos is back to 50 as before and $k_{e_1} > k_{e_2}$. Which was the goal as $v_{s_1} < v_{s_2}$.

5.2 Linear programming model

Let us first briefly mention all the requirements for our model before discussing it in detail. As mentioned before we want to find the largest sum of possible combinations of $c_{s,e}$. Every employee e needs exactly one schedule s. We also have to look at the skills required for certain shifts and last but certainly not least: we want to able to find a feasible solution in any case. The reason for this is that it is better to find an extremely bad roster than none at all.

To start, we make sure every employee is given exactly one schedule. To do this the following sets are used.

E	the set of employees,
S_e	the set of schedules for an employee e

Recall that we don't have to consider any work regulations as we looked at those making the schedules. The start of the model looks like this:

$$\begin{aligned}
\max \quad & \sum_{e \in E} \sum_{s \in S_e} c_{s,e} x_{s,e} \\
\text{s.t.} \quad & \sum_{s \in S_e} x_{s,e} = 1 \qquad \forall e \in E \\
& x_{s,e} \in \{0, 1\}.
\end{aligned}$$

The next step is to make sure that the required shifts are being assigned. To do this we introduce the following.

T	the set of different start and end times of a shift,
G	the set of possible skill sets,
E_g	the set of employees that have exactly skill set $g \in G$.
$q_{t,g} \in \mathbb{N}$	denotes the number of required employees at shift $t \in T$ with skill set $g \in G$.

To determine how many employees got assigned a given shift, $y_{t,s}$ is used.

$$y_{t,s} = \begin{cases} 1 & \text{shift } t \text{ is in schedule } s \\ 0 & \text{otherwise.} \end{cases}$$

To make sure the model is almost always feasible, overstaffing $o_{t,g}$ and understaffing $u_{t,g}$ is introduced. This is done at a large cost O to assure that over- or understaffing won't occur unless it is unavoidable.
The reason that the model won't always be feasible is that it is in theory possible for an employee e to have 0 possible schedules in which case $\sum_{s \in S_e} x_{s,e} = 1$ doesn't hold.
Adding shift requirements and over- and understaffing to the model gives:

$$
\begin{aligned}
\max \quad & \sum_{e\in E}\sum_{s\in S_e} c_{s,e}x_{s,e} - O\sum_{\substack{t\in T,\\ g\in G}}(o_{t,g}+u_{t,g}) \\
\text{s.t.} \quad & \sum_{s\in S_e} x_{s,e} = 1 && \forall e\in E \\
& q_{t,g} = \sum_{e\in E_g}\sum_{s\in S_e} y_{t,s}x_{s,e} - o_{t,g} + u_{t,g} && \forall t\in T, g\in G \\
& x_{s,e}\in\{0,1\} \\
& o_{t,g}, u_{t,g}\in\mathbb{N}.
\end{aligned}
$$

However if an employee has more skills than required for a shift he is also able to fulfill that shift. To add this feature an idea introduced in [9] is used. The sets $\mathcal{F}_{t,g}$ and $\mathcal{G}_{t,g}$ will be defined for every pair (t,g):

$$
\begin{aligned}
\mathcal{F}_{t,g} &= \{g'\in G \mid g'\supsetneq g, \quad \#g'\setminus g = 1\}, \\
\mathcal{G}_{t,g} &= \{g'\in G \mid g'\subsetneq g, \quad \#g\setminus g' = 1\}.
\end{aligned}
$$

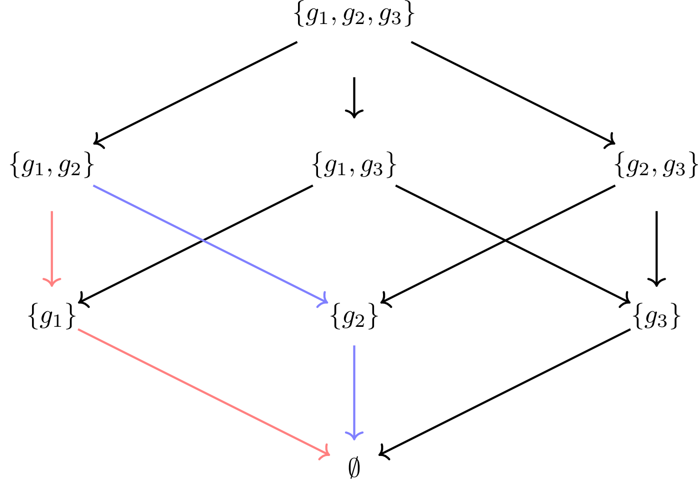
This makes it possible to move down from a skill set g to a subset g' of g which contains exactly one skill fewer, using $z_{t,g,g'}$.

$z_{t,g,g'}\in\mathbb{N}$ the number of employees moving down from g to g' for a shift at $t\in T$.

Example 5.5. In the scenario that there are three different skills g_1,g_2,g_3 and t is fixed this will result in the following sets. For sake of readability the singleton sets $\{g_i\}$ will be denoted $\overline{g_i}$.

$$
\begin{aligned}
\mathcal{F}_{t,\emptyset} &= \{\overline{g_1},\overline{g_2},\overline{g_3}\}, & \mathcal{G}_{t,\emptyset} &= \emptyset, \\
\mathcal{F}_{t,\overline{g_1}} &= \{\{g_1,g_2\},\{g_1,g_3\}\}, & \mathcal{G}_{t,\overline{g_1}} &= \emptyset, \\
\mathcal{F}_{t,\overline{g_2}} &= \{\{g_1,g_2\},\{g_2,g_3\}\}, & \mathcal{G}_{t,\overline{g_2}} &= \emptyset, \\
\mathcal{F}_{t,\overline{g_3}} &= \{\{g_1,g_3\},\{g_2,g_3\}\}, & \mathcal{G}_{t,\overline{g_3}} &= \emptyset, \\
\mathcal{F}_{t,\{g_1,g_2\}} &= \{\{g_1,g_2,g_3\}\}, & \mathcal{G}_{t,\{g_1,g_2\}} &= \{\overline{g_1},\overline{g_2}\}, \\
\mathcal{F}_{t,\{g_1,g_3\}} &= \{\{g_1,g_2,g_3\}\}, & \mathcal{G}_{t,\{g_1,g_3\}} &= \{\overline{g_1},\overline{g_3}\}, \\
\mathcal{F}_{t,\{g_2,g_3\}} &= \{\{g_1,g_2,g_3\}\}, & \mathcal{G}_{t,\{g_2,g_3\}} &= \{\overline{g_2},\overline{g_3}\}, \\
\mathcal{F}_{t,\{g_1,g_2,g_3\}} &= \emptyset, & \mathcal{G}_{t,\{g_1,g_2,g_3\}} &= \{\{g_1,g_2\},\{g_1,g_3\},\{g_2,g_3\}\}.
\end{aligned}
$$

If an employee e with skill set $\{g_1,g_2\}$ gets a shift t that requires skill set $\overline{g_1}$ then $z_{t,\{g_1,g_2\},\overline{g_1}} = 1$. And if e gets the shift t' that requires no skills then either $z_{t',\{g_1,g_2\},\overline{g_1}} = z_{t,\overline{g_1},\emptyset} = 1$ or $z_{t',\{g_1,g_2\},\overline{g_2}} = z_{t,\overline{g_2},\emptyset} = 1$. To elaborate this a graph of the possible paths is given below. In the following graph the possible paths for this example have been colored.



For each pair (t, g) the number of employees moving down can't be larger than the number of employees working the shift plus those coming from higher skill sets. Hence the inequality

$$\sum_{g' \in \mathcal{G}_{t,g}} z_{t,g,g'} \leq \sum_{e \in E_g} y_{t,s} x_{s,e} + \sum_{g' \in \mathcal{F}_{t,g}} z_{t,g',g}$$

has to be added.

This results in the following linear programming model:

$$\begin{aligned}
\max \quad & \sum_{e \in E} \sum_{s \in S_e} c_{s,e} x_{s,e} - O \sum_{\substack{t \in T, \\ g \in G}} (o_{t,g} + u_{t,g}) - \sum_{\substack{g' \subsetneq g, \\ \# g \setminus g' = 1}} (Z_{g,g'} \sum_{t \in T} z_{t,g,g'}) \\
\text{s.t.} \quad & \sum_{s \in S_e} x_{s,e} = 1 && \forall e \in E \\
& q_{t,g} = \sum_{e \in E_g} \sum_{s \in S_e} y_{t,s} x_{s,e} + \sum_{g' \in \mathcal{F}_{t,g}} z_{t,g',g} - \sum_{g' \in \mathcal{G}_{t,g}} z_{t,g,g'} \\
& \qquad - o_{t,g} + u_{t,g} && \forall t \in T, g \in G \\
& \sum_{g' \in \mathcal{G}_{t,g}} z_{t,g,g'} \leq \sum_{e \in E_g} y_{t,s} x_{s,e} + \sum_{g' \in \mathcal{F}_{t,g}} z_{t,g',g} && \forall t \in T, g \in G \\
& \\
& x_{s,e} \in \{0, 1\} \\
& o_{t,g}, u_{t,g}, z_{t,g,g'} \in \mathbb{N}.
\end{aligned}$$

Note that we also added some constants $Z_{g,g'}$. These are added as a cost for assigning an overqualified employee and because of some symmetries that are explained in Section 5.3.1.

Example 5.6. Looking at a shift s' at time t which requires skill g. Recall that we don't look at skills when creating the possible schedules. So it is possible that an employee e that has no particular skills can have schedules in which s' is assigned to him.
If such a schedule s is selected then this will result in overstaffing as $q_{t,\emptyset} = 0$. Because O is large this will only happen if there is no other shifts available for e.

5.3 CPLEX

To solve our model we will use IBM ILOG CPLEX, or CPLEX for short. In 1988 Robert Bixby started development of the simplex method in C, hence the name CPLEX. Throughout the years other methods have been added such as the dual method, the barrier interior method, sifting, mixed integer programming and the possibility to use quadratric programming.[10]
ILOG acquired the project in 1997 and since 2009 it is owned by IBM and is still being developed actively.

After setting the variables, goal and constraints, CPLEX looks for an optimal feasible solution. Because CPLEX uses a lot of heuristics there is no way to describe what happens in detail. But to give some idea we will give a small summary what happens to our integer programming problem.
First off CPLEX will look for reductions in the model to eliminate rows and columns and look for symmetries.
Next it will look for an initial solution. The search method used for this will by default be determined by CPLEX. The method chosen depends on different factors. E.g, size, column/row ratio etc.
After this the solution will be improved by using for example branch and bound which is explained in Section 3.3.
After a time limit or iteration limit set prior to the run CPLEX will halt and return the best solution found and if applicable whether it is optimal.

5.3.1 Improvements

Subproblems:
Recall from Section 3.1 that integer programming in general is an NP-problem. To tackle this, the problem is divided into smaller subproblems and those are solved and combined to come to a final solution.
In our case this means that we will take a subset of the schedules and solve the puzzle. After this a new subset is selected in which the schedule of the previous solution is included and then the new problem is solved. By including the previous solution we guarantee that the solution won't be worse compared to previous runs.

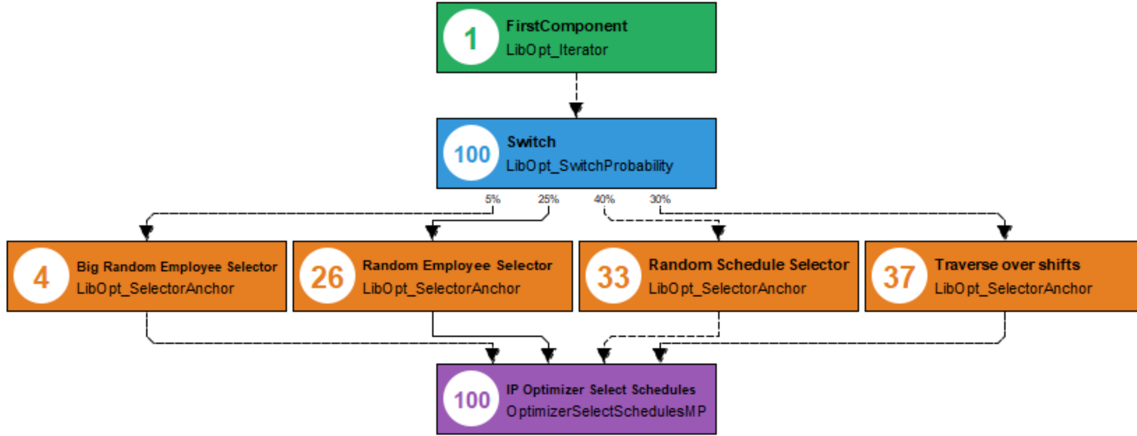


Figure 4: An example of subproblems and how often they were chosen.

The `Random Employee Selector` selects one employee and selects a number of schedules that are possible for that employee at random. In our case this was 1000. Because the schedules are made for a category this means that the schedule-employee combination will be created for every employee that is in the same category as the chosen employee.
The `Big Random Employee Selector` does exactly the same but uses more schedules. Using more schedules will result in a slower process, but also in a more accurate solution. To make sure we don't lose too much time in this process this selector is used less often.
The `Random Selector` will pick some schedules at random. For each schedule all the schedule-employee combinations are added for all employees in the category of that schedule.
We will also `Traverse over shifts` by selecting schedules that contain a certain shift. The reason behind this is that every shift must have sufficient employees and random schedules might miss some hard to plan shifts. For example shifts that can be assigned to only a few employees due to skill restrictions.

To create a better optimizer one must experiment with a lot of variables: the number of schedules can be higher or lower, the distribution over the different methods can be changed and other subproblems can be made and tested. E.g, traverse over employees, chose schedules depending on their value, chose an employee depending on the amount of kudos, etc.

Removing redundant constraints:
An other improvement that was tried, is removing the redundant constraints on the shifts. When this is allowed is illustrated by the following example.

Example 5.7. Let t be a shift and $G = \{g_1, g_2, g_3\}$ with $q_{t,\emptyset} = 1$ and $q_{t,g} = 0$ otherwise. It isn't interesting to consider $2^3 = 8$ constraints if only one is required to be nonzero and removing the other constraints may seem interesting.
However one should be careful with this. Employees should be considered as well. Let e be an employee with skill set $\{g_1, g_2\}$. Recall that the sum $\sum_{e \in E_g} y_{t,s} x_{s,e}$ is used when determining $q_{t,g}$. If $q_{t,\{g_1,g_2\}}$ is removed then the shift t for employee e will not be take into account. However this is important as e is allowed to work shift t.

So $q_{t,\{g_1,g_2\}}$ is kept. Together with that also $q_{t,\overline{g_1}}$ and $q_{t,\overline{g_2}}$ aren't removed to make it possible for employee e to scale down to $q_{t,\emptyset}$.
Assuming there are no employees with other skill sets the other constraints can be removed.

So a constraint for $q_{t,g}$ can be removed if for all $g' \supseteq g$ we have $q_{t,g'} = 0$ and there are no employees with skill set g'.
When implementing this and comparing the results with previous runs no significant differences in time or optimal solution were noticed. However the number of rows removed by CPLEX decreased. This is of course due to the fact that they weren't added in the first place.

Removing symmetries:
There are two different symmetries that we have tried to tackle. These will be explained using an example.

Example 5.8. Let there be an optimal solution with overstaffing at shift t for an employee e with skill g. So $o_{t,g} = 1$. Assume for now that $Z_{g,g'}$ has not yet been introduced. Another solution with $o_{t,g} = 0, z_{t,g,\emptyset} = 1$ and $o_{t,\emptyset} = 1$, and other variables equal to the previous solution, will also be a feasible solution.
The constraint concerning $q_{t,q}$ is still satisfied as $o_{t,g}$ and $z_{t,g,\emptyset}$ balance out the changes. The same goes for $q_{t,\emptyset}$ where $z_{t,g,\emptyset}$ and $o_{t,\emptyset}$ balancing each other out. The inequality concerning $z_{t,g,\emptyset}$ isn't violated either as $y_{t,s}x_{s,e} \geq 1$, because otherwise there would be no overstaffing. Other constraints are not altered.
Both solutions have the same value as O is independent of the shift and skill set and $z_{t,g,\emptyset}$ does not contribute to the solution value.

To overcome this symmetry a positive value $Z_{g,g'}$ is introduced. For this particular case the values can be equal for all g, g', as long as $Z_{g,g'} > 0$. The next example, explaining the second case of symmetry, shows why these constant $Z_{g,g'}$ may not be good enough.

Example 5.9. Let there be an optimal solution and an employee e with skill set $\{g_1, g_2\}$ working shift t requiring no skills. Assume that $z_{t,\{g_1,g_2\},\overline{g_1}} = z_{t,\overline{g_1},\emptyset} = 1$. If no $Z_{g,g'}$ is introduced or if $Z_{g,g'}$ is equal for all g, g' the following solution has the same value: $z_{t,\{g_1,g_2\},\overline{g_1}} = z_{t,\overline{g_1},\emptyset} = 0$ and $z_{t,\{g_1,g_2\},\overline{g_2}} = z_{t,\overline{g_2},\emptyset} = 1$ and other values the same as before.
Because of the values of $z_{t,\{g_1,g_2\},\overline{g_1}}$ and $z_{t,\{g_1,g_2\},\overline{g_2}}$ the constraint for $q_{t,\{g_1,g_2\}}$ is still satisfied. Because $z_{t,\{g_1,g_2\},\overline{g_i}} = z_{t,\overline{g_i},\emptyset}$ for both i for both solutions the constraints for $q_{t,\overline{g_i}}$ aren't violated either and because of the values of $z_{t,\overline{g_1},\emptyset}$ and $z_{t,\overline{g_2},\emptyset}$ the constraint concerning $q_{t,\emptyset}$ still holds. Also none of the inequalities are violated either. Therefore two different feasible solutions can be found with equal value.

To create a unique optimal solution the $Z_{g,g'}$ should be defined such that each possible path from a skill set g to g' with $g' \subseteq g$ should have a unique path of minimum value.
Let each skill g_i have a value $v(\overline{g_i}) = i$ and define $Z_{g,g'} = \#g \cdot v(g \setminus g')$. Testing this, up to 9 skills a path with a unique minimum exists for each pair g, g' with $g' \subseteq g$.
However implementing this in CPLEX caused the program to slow down tremendously. The better solution was to let CPLEX find the symmetry rather than trying to remove it.

Sifting settings:
The last thing that was tried was adjusting the setting for sifting in CPLEX. The sifting optimizer is used when there are significantly more variables than constraints. Denoting g for the number of skills, s the number of schedules and t the number of shifts. Then the number of variables is

$\mathcal{O}(s \cdot e + t \cdot 2^g + t \cdot 2^g \cdot g)$ and the number of constraints $\mathcal{O}(e + t \cdot 2^g)$.
As s is significantly larger than e, t and g the number of variables is also significantly larger than the number of constraints.
For this reason the solving method in CPLEX for the sifting optimizer was changed from default to yes. However this caused a reduction in speed. CPLEX already used sifting when required, but forcing it in other cases caused the speed to decrease.

6 Tests

There are three points that are important to be tested. First of all we want to create the correct possible schedules. It is also important to know the speed of the algorithm and number of schedules that are made depending on the number of shifts. After this the model will be used on a small scale to test calculation of the kudos. For this we look at several ways to update and scale the kudos and the combinations of those. Finally the model itself will be checked on a larger scale, i.e, with more employees.

6.1 Possible schedules

Recall that for creating the possible schedules it is not important how often a shift appears or which skills are required. There will be several **different categories** that will be tested.
These are based on certain differences that might result in a difference in the number of possible schedules found for a category. For this reason we check two different contract lengths: 20 hour and 38 hours. A category either has no days off or Thursday off. It has either no holiday or a holiday on Friday. The preference for a day off can be none or Wednesday and finally the preference to have a part of a day off could be in the morning or empty.

As the preferences to work only affects the value of the schedule and not how they are made, we will not consider those preferences. The night shifts are set to true because we want to look at all the shifts at all times. Almost all the combinations will be considered which are $2^5 = 32$ different categories. However if the only prefer to have a day off the maximum violations of 3 will never be violated because every day will have at most one shift, and thus the maximum number of violations will be at most one. This results in the same category with no day off preferences having the same number of possible schedules. Therefore the preference to have one day off will only be considered in combination with the preference to have a part of a day off.

The possible schedules will be created for all of the categories. With each step we will add a shift on each day of the week on the same time. The shifts will be added in the following order to keep them spread over the day:

1. 0:00–8:00
2. 8:00–16:00
3. 16:00–0:00
4. 4:00–12:00
5. 12:00–20:00
6. 20:00–4:00
7. 2:00–10:00
8. 6:00–14:00
9. 10:00–18:00
10. 14:00–22:00

As seen before: when determining the time complexity we saw that removing double schedules could take a lot of time. That's why we are not only interested in the total time, but also in the time spent removing duplicates.

Number of shifts	1	2	3	4	5	6	7	8	9	10
Create	0,13s	0,24s	0,86s	2,66s	6,77s	14,4s	28,6s	51,0s	1:09m	2:00m
Remove duplicates	0,04s	0,08s	0,34s	0,74s	1,73s	4,7s	8,8s	36,0s	1:05m	2:54m
Total time	0,17s	0,32s	1,2s	3,4s	8,5s	19,1s	37,4s	1:27m	2:14m	4:54m

Figure 5: The time it took to create possible schedules for all categories.

A full overview of the number of schedules can be found in Appendix A. Some categories are pointed out here to give an idea of an upper and lower bound and the speed at which the number of schedules increases.

(a) 38 hours contract without any restrictions

(b) 38 hours with all possible restrictions given in the first paragraph of this section.

(c) 20 hours contract without any restrictions

(d) 20 hours with all possible restrictions given in the first paragraph of this section.

Category \ Number of shifts	1	2	3	4	5	6	7	8	9	10
(a)	6	56	252	1960	6174	12144	28408	67998	120552	195456
(b)	2	21	96	372	1302	2787	4576	7242	15596	28579
(c)	55	290	824	2013	3844	6260	9908	15087	21268	28786
(d)	25	109	286	637	1204	1944	2902	4140	5960	8176

Figure 6: The number of possible schedules for a selection of categories.

6.2 Kudos

It is important to test the behaviour of the kudos in many different situations to check for unexpected results. For each test we will give a reasoning why we do this test and what sounds like a fair result before comparing those to the test results. This way we find a way to update the kudos in a fair way for all the employees.
Each test will start with all employees having an equal amount of kudos, and continues for several weeks. This way we can see what the kudos will do in the long run.
In general there are several different variables to test which are briefly mentioned again. A more elaborate explanation can be found in Section 5.1.

1. Creating the value of the schedule $c_{s,e}$ depending on schedule value v_s, kudos k_e and s' the number of shifts in s:
   (a) $v_s + k_e$;
   (b) $v_s \cdot k_e$;
   (c) $\frac{v_s}{2s'} \cdot k_e$;
   (d) $(\frac{v_s}{2s'})^2 \cdot k_e$;
   (e) $\sqrt{\frac{v_s}{2s'}} \cdot k_e$.
2. Recalculating the kudos depending on the value of the chosen schedule v_s:
   (a) $k_e - v_s + s'$;
   (b) $k_e \cdot \frac{s'}{v_s}$;
   (c) $\begin{cases} k_e \cdot \frac{s'}{v_s} & v_s \geq s' \\ k_e \cdot (-\frac{v_s}{s'} + 2) & v_s < s'; \end{cases}$
   (d) $k_e \cdot \frac{m_s}{v_s}$.
3. Scaling the kudos to keep the average equal:
   (a) additive;
   (b) multiplicative.
4. Average kudos at all times:
   (a) 20;
   (b) 50;
   (c) 80.

To test every combination of the variables above each test has to be run 90 times. This would take too much time, so several combinations have been chosen to start with. Throughout the tests some changes are made to the used variables and if something doesn't seem to work it won't be tested any more than necessary.

Keep in mind that if an employee works 38 hours all schedules will contain 5 shifts and if an employee works 20 hours they will contain either 2 or 3 shifts.

6.2.1 Does it select preferences?

Before starting to test all sorts of borderline cases there should be a check on the validity in general. Take employees $e_1, \ldots, e_7$ that work full time and each employee e_i the preference to have a day off on day i.
Each day will consist of 5 shifts spread over the day. These are added in the same order as with the test of the possible schedules. The $c_{s,e}$ are defined by 1(b). Every combination of recalculation and scaling of kudos will be tested.
Expected is that every shift will be assigned and the preferences can be met. If every preference can be met there is no expected change in the amount of kudos and in that case testing the different ways to recalculate and scale becomes redundant.

Results:

	Monday					Tuesday					Wednesday					Thursday				
	0:00	4:00	8:00	12:00	16:00	0:00	4:00	8:00	12:00	16:00	0:00	4:00	8:00	12:00	16:00	0:00	4:00	8:00	12:00	16:00
e_1											■									■
e_2												■				■				
e_3	■						■													
e_4		■				■														
e_5			■							■					■				■	
e_6				■				■					■				■			
e_7					■				■					■				■		

	Friday					Saturday					Sunday				
	0:00	4:00	8:00	12:00	16:00	0:00	4:00	8:00	12:00	16:00	0:00	4:00	8:00	12:00	16:00
e_1					■				■				■		
e_2			■				■								■
e_3	■					■					■				
e_4				■						■				■	
e_5														■	
e_6												■			
e_7		■													

After each run we get a roster in which every employee e_i doesn't have to work on day i. However there is one overstaffing and one understaffing every time. After making a roster without the preferences the same amount of over- and understaffing occurs. So this wasn't caused by the preferences of the employees, but due to the restricting work regulations.
As mentioned above the recalculation and scaling does not require any testing in this scenario.

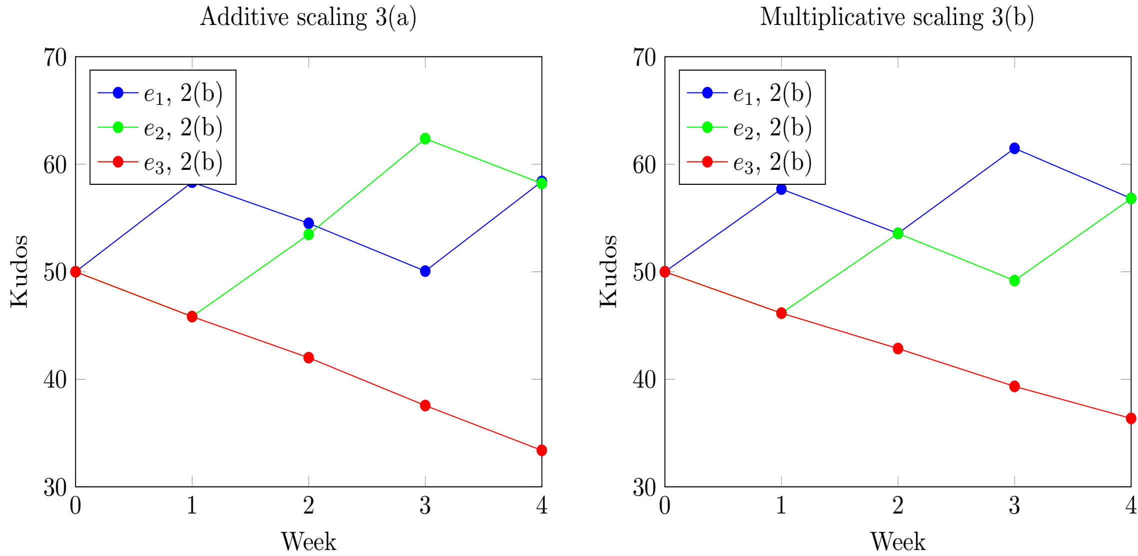
6.2.2 Prefer to have a day off on the same day

During this test we will have employees e_1, e_2, e_3 all working full time. The employees e_1 and e_2 prefer Tuesday off, e_3 has no preferences. There will be 2 shifts each day. This way we expect that e_3 will always work on Tuesday and either e_1 or e_2 has one assigned shift on Tuesday. It is also expected that whether e_1 of e_2 works on Tuesday will alternate. The goal is that after every

two weeks the amount of kudos is back to the starting point, because both employees got the same number of preferred shifts.

In the first run we compare the difference in behaviour in 3(a) and 3(b), scaling of kudos, while using 1(b), 2(b) and 4(b).

Results:



Looking at the graph on the left we can see that e_2 got his preferences in the first week by the decrease in kudos. The second week e_1 got his preferences which can again be seen by the drop in kudos. This is what was expected as the kudos were higher. The third week e_1 gets his preference again, as the kudos is still slightly higher. Employee e_2 is compensated for this with an increase in kudos and got his preference again in week 4.
The same happens in the graph on the right except that after two weeks the kudos is again equal so the employee that will get his preference first is random as in the first week.
Running the same tests with 2(a) and 2(c) give comparable results which can be found in Appendix B.1.

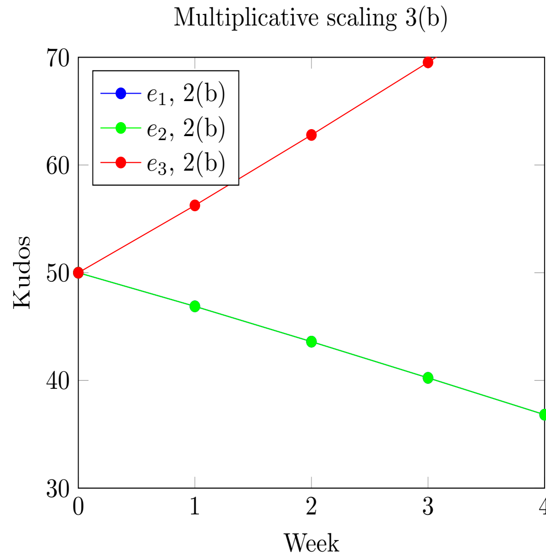
Multiplicative scaling seems to give slightly better results, as additive scaling is slightly off for 2(a) and 2(c).
Note that the kudos of employee e_3 without preferences has been going down every time. Which isn't that surprising, since his schedule has no bad shifts. However this doesn't seem fair and will be addressed shortly.

6.2.3 Prefer to work on the same day

The test is exactly the same as before except that employees e_1 and e_2 want to work on Tuesday instead of having a day off. It is expected that both will get their preferences as there are two shifts

and that this would result in keeping the amount of kudos of those two employees the same. At the same time it is expected that the amount of kudos of e_3 increases.
Because we are mostly interested in the behaviour of the kudos of e_3 right now the test will only be done for 1(b),2(b),3(b) and 4(b).

Results:



The results are as expected and the way kudos for employees without preferences should be addressed.

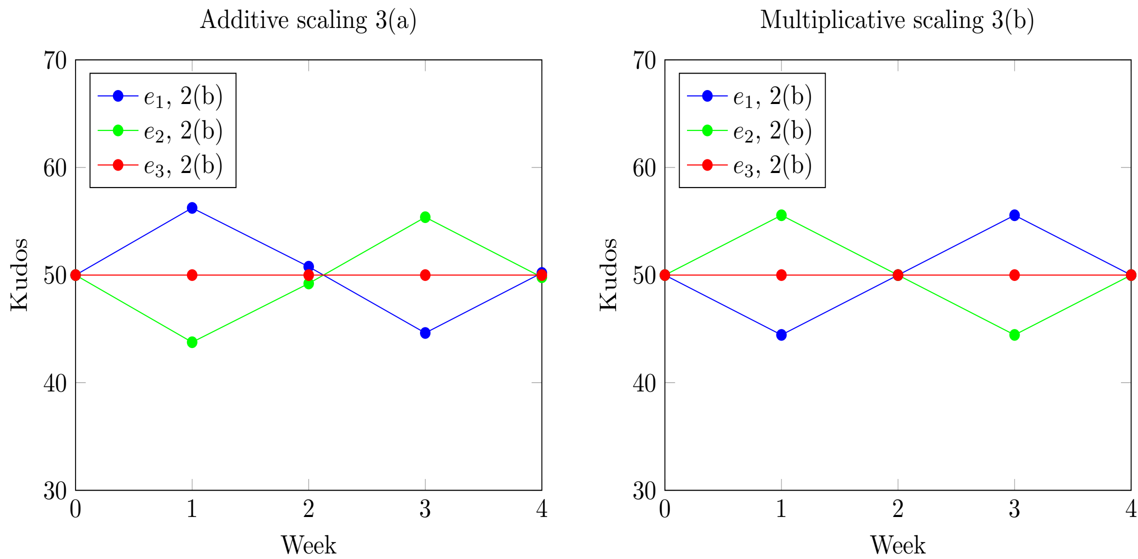
6.2.4 Concerning employee without preferences

This isn't a test, but an important change that is made between the previous tests and the tests yet to come. Because the change of kudos of an employee without preferences is in general not predictable we will no longer change those. The previous tests will be reviewed to make sure the results are still similar for the other employees before continuing with the next tests.

6.2.5 Prefer the same day off/work revisited

The tests where both employees prefer to work isn't very interesting. Employee e_1 and e_2 get equivalent schedules and the kudos don't change. All three employees keep the average kudos after every week.
When both e_1 and e_2 prefer to have a day off we check the combinations of recalculating and scaling while keeping 1(b) and 4(b) fixed. The ideal result would be that both employees get their preference once and that the kudos are back to the starting point after two weeks.

Results:

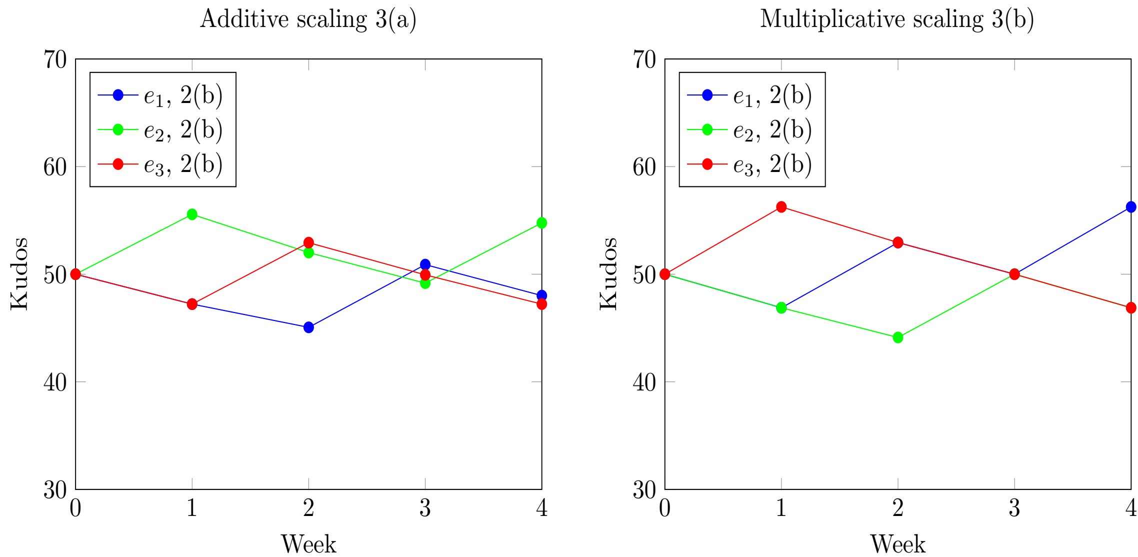


In every test both e_1 and e_2 got their preference twice. In most cases the kudos did also return to their exact start position. However, as seen in the graphs, the method 2(b) doesn't go together well with 3(a). This is most likely the result of using multiplication when updating the kudos but use addition when the kudos are being scaled. The small deviation is also seen with 2(c) and 3(a). All the results are shown in Appendix B.2.

6.2.6 Conflict with three employees

This time there will be three employees e_1, e_2, e_3 that all would like to work on Tuesday. As before we will set two shifts per day, which means that not all three employees can have their preference. The solution to this would to let everyone work on Tuesday two out of three times and after three weeks be back at the starting point. As before we test the recalculation and scaling while using 1(b) and 4(b).

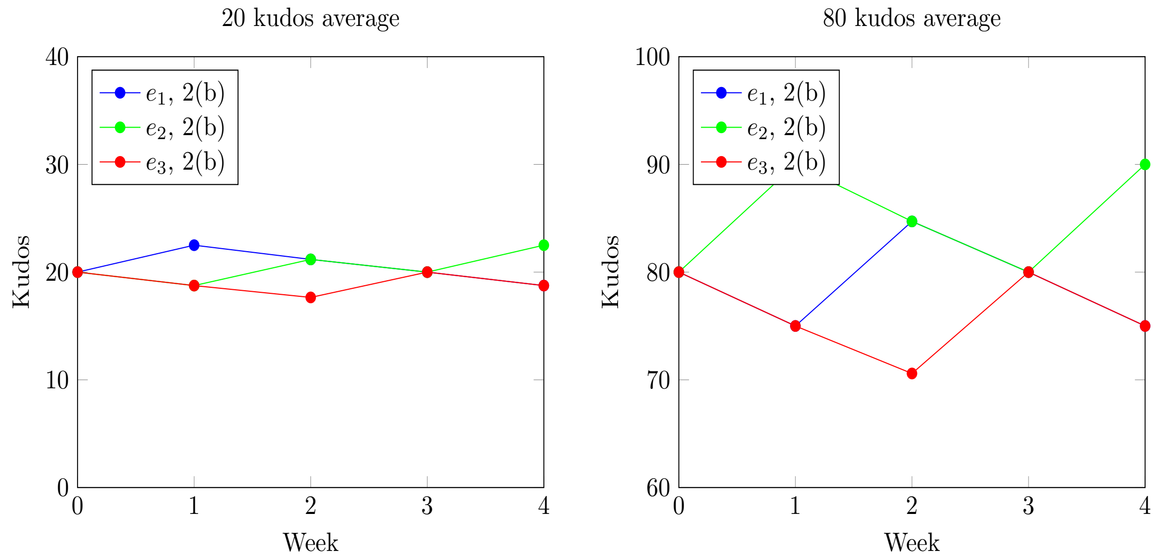
Results:



In every run all three employees got to work on a Tuesday twice in the first three weeks. This is also reflected in the graphs: the amount of kudos drops twice for every employee. However we didn't get back to the exact starting point in every run. Once again this didn't occur for 3(a) combined with 2(b) and 2(c) For this reason we will continue testing with 3(b). All the results are shown in Appendix B.3.

Test kudos average:
This test is also used to test the variable of 4, the average amount of kudos. The idea behind varying the average is that for two schedules, one equally bad as the other is good, the relative distance to the average can be different. To check this we fix 1(b) and 3(b) and vary 2 and 4.

Results:

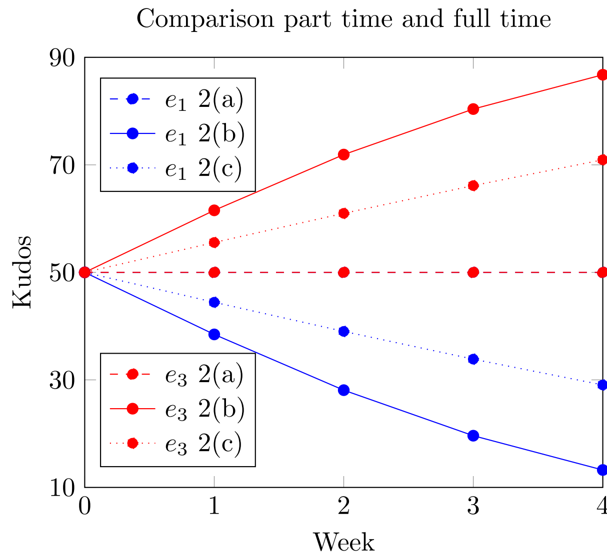


Comparing all the graphs found in Appendix B.3, we see that there are no changes for 2(a). For 2(b) and 2(c) the amount of kudos added or removed differ but these don't lead to differences in making the rosters. So the average will not be tested anymore.

6.2.7 Behaviour with different contract hours

Take four employees $e_1, \ldots, e_4$. Employees e_1 and e_2 work 38 hours, the others work 20 hours. e_1 and e_3 prefer Tuesdays off, e_2 and e_4 have Tuesdays off. There are two shifts each day which shouldn't result in overstaffing if every part time employee works two shifts. So we expect neither e_1 nor e_3 to ever get their preferences. However we are interested in the amount of kudos that are lost when employees have different contract hours.
We did this test keeping 1(b), 3(b) and 4(b) fixed and varying 2, the recalculating of kudos.
In the graphs only two employees are plotted. As e_2 and e_4 have no preferences their kudos will remain constant.

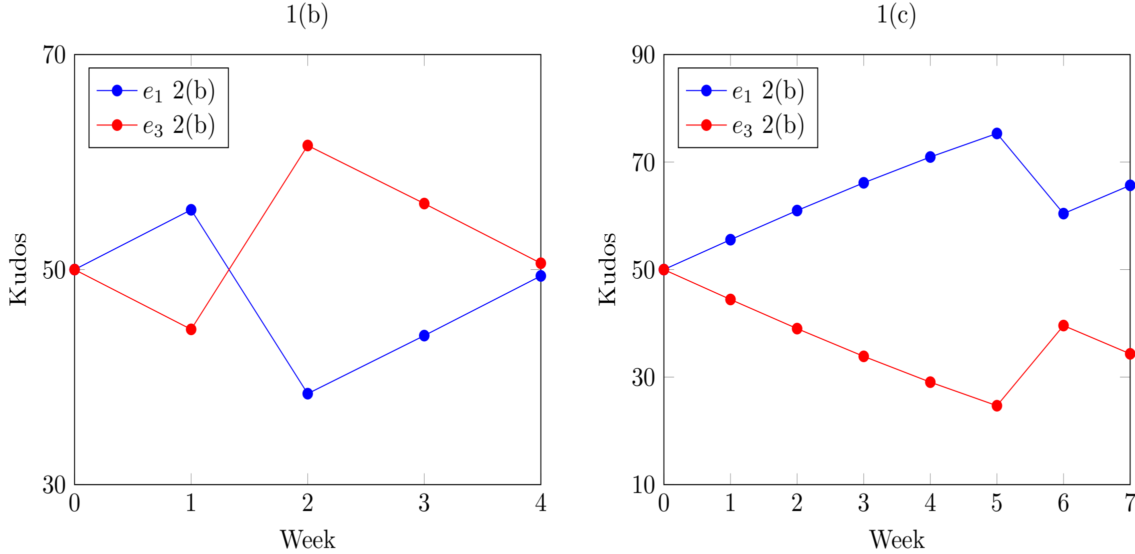
Results:



When using 2(b) and 2(c) we see an increase in kudos for part time employee e_3. The reason is pretty clear: if you have fewer shifts to compensate for one bad shift the schedule will be worse. 2(a) remains equal because this considers shifts absolute instead of relative: 50-4+5=50-2+3.

Testing when not all preferences can be satisfied:
One may wonder why we picked four employees for the previous test. The reason for this is that we will run the same test but this time e_2 doesn't have a day off so he will be assigned one shift on Tuesday. In general we hope that both employees will get their preferences but that e_4 would get it more often. Once again we fix 3(b) and 4(b), but 1, valuing $c_{s,e}$, and 2, recalculating kudos, will vary.

Results:



The other results can be found in Appendix B.4.

1(a): This option doesn't work. An example is given to explain why.

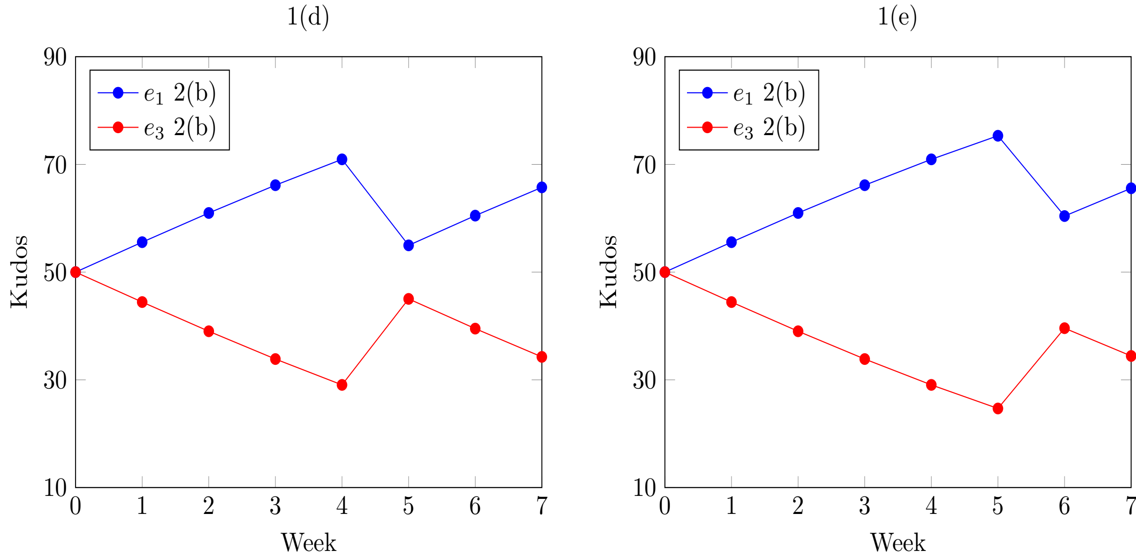
Example 6.1. Let e_1, e_2 be employees with the same preference and only one preference can be fulfilled each week, like during this test. To determine who gets which schedule we maximize $v_{s_1} + k_{e_1} + v_{s_2} + k_{e_2}$ where s_i is the schedule for employee e_i.
Because the kudos are constants when selecting schedules we are just maximizing $v_{s_1} + v_{s_2}$ and this is independent of the amount of kudos an employee has. So, because the employees have the same preferences, the solution assigning s_1 to e_2 and s_2 to e_1 is also optimal. Multiplying by some factor to maximize over, for example, $c_v v_{s_1} + c_k k_{e_1} + c_v v_{s_2} + c_k k_{e_2}$ or using other methods like the square, $v_{s_1}^2 + k_{e_1}^2 + v_{s_2}^2 + k_{e_2}^2$ doesn't change anything to this fact.

1(b): When applying 1(b) it is important to note that when the kudos are equal the first schedule will be assigned random and there is no advantage for part time or full time as $c_{s,e} = v_s \cdot k_e$ doesn't depend on the number of shifts. What we see in the graph is that if the kudos of the full time employee e_1 drops, it drops more than for the part timer e_3. This is not the case for 2(a), which doesn't depend on the number of shifts. The slower decrease of kudos results in the part time employee getting his preference more often.
1(c): Looking at the graph you can see that there is a certain height of kudos a full time employee must have before getting his preference over a part time employee. To determine this amount we solve

$$\frac{4}{10}x + \frac{2}{4}(100 - x) = \frac{5}{10}x + \frac{1}{4}(100 - x)$$

for x. Which is the point where both employees are as likely to get their preference. This results in $x \approx 71.43$. 2(a) will achieve this height very slowly as the kudos can't change much in one week. We see that 2(b) achieves this in week 5 and drops down to a value comparable to week 2. So one

can assume that in for the next three weeks the part timer will get his preference again before the full time employee will get his preference again.
Looking at 2(c) you can see that the drop after week 6 is much less. It drops back to a height similar to week 4. So we expect to get the part time employee his preference twice before the full time employee getting his preference again.



1(d) and 1(e): The results off 1(d) and 1(e) are very similar to 1(c). The speed at which the kudos increase and decrease are independent of 1, the way we value the schedules. The only difference is the height at which point the full time employee gets his preference over the part time employee. This his slightly lower when using 1(d), the quadratic method, and slightly higher when using 1(e), the square root method. For 1(d) we solve

$$(\frac{4}{10})^2 x + (\frac{2}{4})^2(100 - x) = (\frac{5}{10})^2 x + (\frac{1}{4})^2(100 - x)$$

which gives $x \approx 67.57$. For 1(e) we solve

$$\sqrt{\frac{4}{10}x} + \sqrt{\frac{2}{4}(100 - x)} = \sqrt{\frac{5}{10}x} + \sqrt{\frac{1}{4}(100 - x)}$$

which results in $x \approx 73.51$

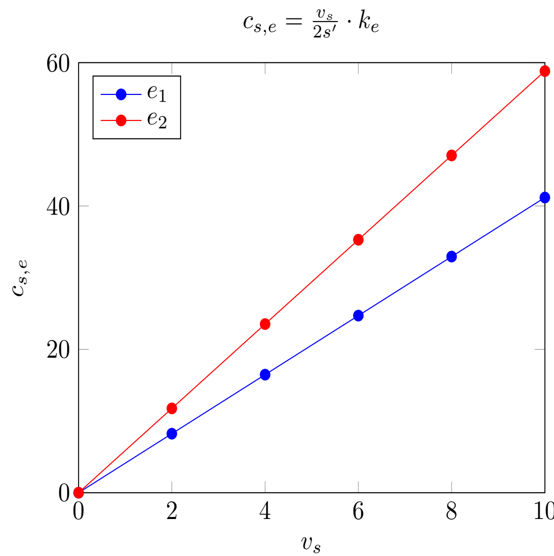
6.2.8 Lots of conflicting preferences

It might occur that there are two employees that want a lot of the same shifts. During this test there are full time employees $e_1, \ldots, e_4$. Employees e_1 and e_2 both prefer to be working during evening shifts. Each day of the week there will be three shifts, one of which is an evening shift. So there are 7 evening shifts to distribute over the two employees. A fair result is that they are distributed as 3 to 4.

Note that there are four employees to create some more freedom in the roster. If only two employees are active we are very restricted because of the working regulations.
This time combinations 1(c)-(e) and 2(a) and 2(c) are considered. 1(b) is skipped because it is the same as 1(c) if all employees have the same contract hours. 2(b) is skipped because $v_s \geq s'$, with s' the number of shifts. In this case 2(b) and 2(c) are the same.

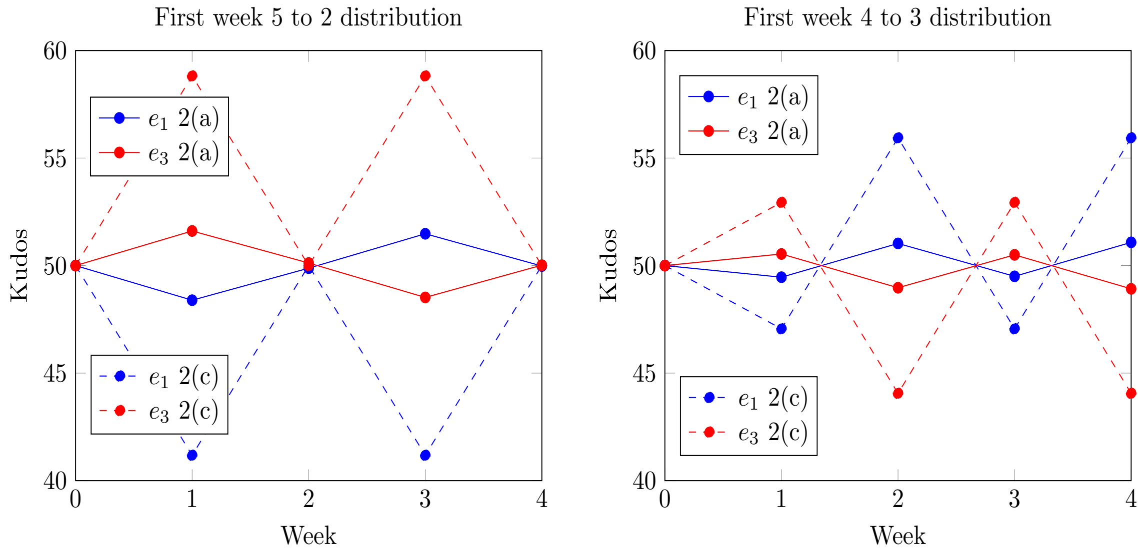
Results:
1(c): The distribution of the 7 evening shifts is random for the first week. The amount of kudos is equal for both employees and the increase is linear. This means that the shifts are assigned 5/2 or 4/3. The next week the distribution is always 5/2. The reason for this is explained with an example.

Example 6.2. Assume we assign 5 evening shifts to e_1 and 2 evening shifts to e_2 will give e_1 41.18 kudos and e_2 58.82 if we recalculate using 2(c). Then the following graph displays the value of a schedule for the next week.



It wouldn't be beneficial to assign 4 shifts to e_2 and 3 shifts to e_1 as the loss of 1 shift for e_2 is considered worse than the gain of 1 shift for e_1. In this case we would compare 5 shifts, $v_s = 10$, and 2 shifts, $v_s = 7$, to 4 shifts, $v_s = 9$, and 3 shifts, $v_s = 8$. As $1 \cdot 58.82 + 0.7 \cdot 41.18 = 87.646 > 85.882 = 0.9 \cdot 58.82 + 0.8 \cdot 41.18$ it is more beneficial to assign the shift to employee e_2.
This difference in kudos may be smaller or larger depending on how the recalculation occurs. However in any case this effect occurs.

In these graphs the different scenarios for employees with preferences are plotted.

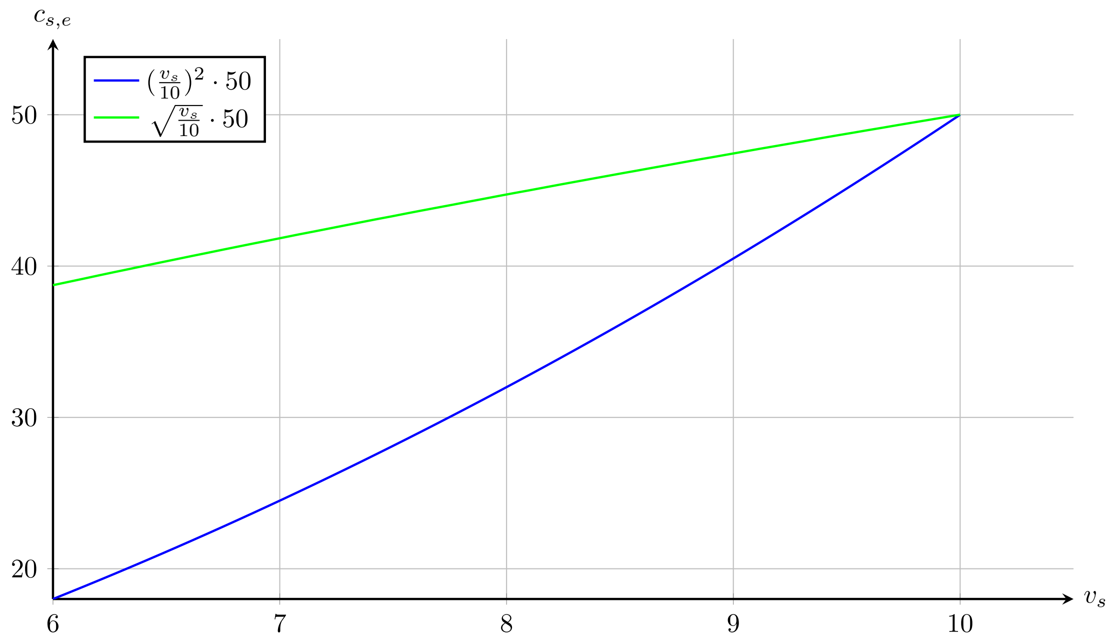


We see that with a 5 to 2 distribution the kudos are back to the starting point for 2(c) and close to the starting point for 2(a). For a 4 to 3 distribution the kudos will never return to a starting point. However there is a repeating pattern for 2(c), and 2(a) is again close to being the same.

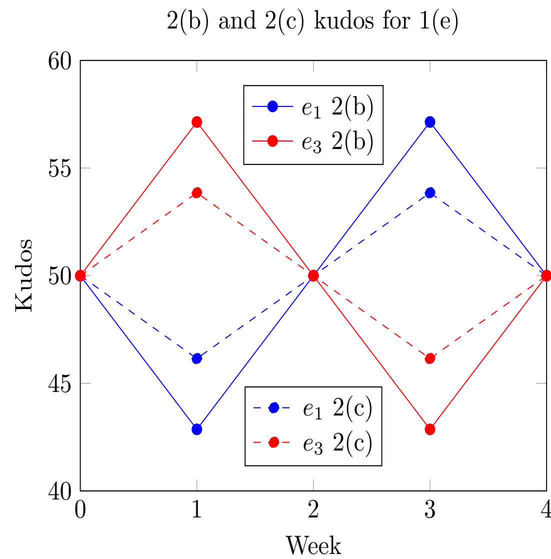
1(d): For the square method the distribution is always 5 to 2. The behaviour of the kudos is similar to previous results. When using 2(c) they return to the starting situation and for 2(a) they are close to the starting situation.

1(e): Using the square root has only two differences compared to the quadratic method. The first is that the distribution is always 4 to 3. For this reason the second difference occurs, which is the amount of kudos an employee has after one week. However after two weeks they are again equal for 2(c) and close to equal for 2(a).

The reason that this difference in distribution occurs is similar to what happens in Example 6.2. Looking at the graph below we see can see that for the square taking the sum of the values $v_s = 8$ and $v_9 =$ is lower than for $v_s = 7$ and $v_s = 10$. So a the shifts are assigned 5 to 2. The exact opposite is true for the square root, which is a why the shifts are assigned 4 to 3.



Prefer not working evening shifts:
To complete this test the variant where employees e_1 and e_2 don't want to work in the evening is considered. This should cause a difference in 2(b) and 2(c).
The results will be considered shortly as they are mostly the same as the working preference in the evenings. The distribution per method of 1, setting $c_{s,e}$, is the same and the amount of kudos after two weeks are the same.
The only difference occurs in the amount of kudos after one week. A comparison of 2(b) and 2(c) using 1(e) is given below. However this has no consequences for how and how often shifts are assigned.



6.3 Model

To test the full process two tests are used. A smaller test has 16 employees, a larger one 48. The data is fictional, but based on real data. While running the tests, the variables that were tested in the previous section will be chosen and kept the same for the upcoming tests. The settings used are as follows: 1(e), using the square root to determine the value of a schedule. 2(b), using the ratio of number of shifts to the value of the schedule. 3(b), scaling multiplicative. 4(b), an average of 50 kudos. Each test will be separated in three parts:

1. Creating a roster with the employees and shifts.
2. Same as 1. but with added skills to employees and shifts.
3. Same as 2. but with added rules such as days off and added preferences.

Because the preferences don't play a role in the first two parts we won't consider kudos until the last part of each test.

6.3.1 16 employees

Following tables give the input of the first run. First the number of employees required for a shift is given and second the employees that are available.

Time \ Day	Monday	Tuesday	Wednesday	Thursday	Friday	Saturday	Sunday
6:00–14:00	5	6	6	5	5	3	2
14:00–22:00	4	5	5	4	4	3	3

Full time	Part time
Al Baird	Naomi Walls
Deb Smith	Nicole Holmes
Isaac Haynes	Oliver Curtis
Jess Lester	Rachel Gold
Leslie Woods	Richard Stone
Lorn Olson	Ricky Rookie
Lorraine Jennings	Rita Sanders
Meredith Miles	Rob Lewis

The results of the first test are promising. A roster with with no over- or understaffing is found every time. An example of the results can be found in Appendix C.1.

For the second part there will be skills added. Every day one of the shifts of 06:00–14:00 will require Chinese as a skill and one of the shifts 14:00–22:00 will require English as a skill. We expect that at least one of the employees working that shift has the required skill.
These are the employees who have skills:

English	Chinese
Isaac Haynes	Jess Lester
Meredith Miles	Meredith Miles
Rita Sanders	Rachel Gold
	Rob Lewis

For the second part we also find rosters that have no over- or understaffing, and on top of that at least one of the eligible employees is assigned to a shift with his skill. An example can be found in Appendix C.1.

The third part will contain preferences and therefore we will also check the updating of the kudos.
Days off: Richard Stone and Ricky Rookie on Wednesday, Nicole Holmes and Rita Sanders on Friday, Oliver Curtis on Saturday and Sunday.
Preferred days off: Rachel Gold and Richard Stone on Tuesday.
Preferred working part of day: Al Baird and Leslie Woods in the morning, Lorraine Jennings and Meredith Miles in the evening.

The roster found can't be better. There is again no over- or understaffing, the skills are assigned correctly and everyone got their preferences. The full roster can been found in Appendix C.1.
However what happens with the kudos requires some explanation. Below is a table of the results of the employees that have preferences.

Employee	Kudos before	Number of shifts	Value of schedule	Kudos after
Al Baird	50	5	10	37.5
Leslie Woods	50	5	10	37.5
Lorraine Jennings	50	5	10	37.5
Meredith Miles	50	5	10	37.5
Rachel Gold	50	2	2	75
Richard Stone	50	3	3	75

It seems fair that if every employee has gotten their preference no change in kudos occurs. However this is not the case. The reason that this happens is that different preferences cause difference in the value of a schedule. For example preferring to work a certain shift increases the value of a schedule, but preferring not to work, only causes the value not to drop. When recalculating the kudos the maximum value of $2s'$ can't always be achieved. In an attempt to make this more fair the maximum value of the possible schedules can be considered, giving the following results.

Employee	Kudos before	Number of shifts	Value of schedule	Kudos after
Al Baird	50	5	10	46.15
Leslie Woods	50	5	10	46.15
Lorraine Jennings	50	5	10	46.15
Meredith Miles	50	5	10	46.15
Rachel Gold	50	2	2	69.23
Richard Stone	50	3	3	46.15

In this scenario we see an increase in the kudos of Rachel. This is caused due to the fact that Rachel has possible schedules containing 3 shifts. So she is the only employee that didn't get the maximum value. However for the number of shifts that are in her schedule the maximum value is achieved. Considering the maximum value for the number of shifts in a schedule the following results are obtained.

Employee	Kudos before	Number of shifts	Value of schedule	Kudos after
Al Baird	50	5	10	50
Leslie Woods	50	5	10	50
Lorraine Jennings	50	5	10	50
Meredith Miles	50	5	10	50
Rachel Gold	50	2	2	50
Richard Stone	50	3	3	50

Note that these settings don't change the value of the roster we find, just the way the kudos are recalculated from $\frac{s'}{v_s} \cdot k_e$ to $\frac{m_s}{v_s} \cdot k_e$. The behaviour of this new method should be tested as well. However, because all the tests of kudos in the previous section are with employees from the same categories, and thus the same possible schedules, these changes don't alter anything between these employees.

6.3.2 48 employees

The following table gives the input of the required employees for shifts.

Day / Time	Monday	Tuesday	Wednesday	Thursday	Friday	Saturday	Sunday
0:00–8:00	8	10	10	8	8	7	7
8:00–16:00	12	14	14	12	12	11	11
16:00–0:00	10	13	13	10	10	10	10

Because of the high number of employees, we will only mention those that stand out. To validate the results it is important to know that the employees working full time have a first name that start with A-M, all part timers have first names that start with N-Z.

Full time	Part time
40	8

During the first test rosters without over- or understaffing are found. A result, which also satisfies the second and third test, is found in Appendix C.2.
The next step is to add skills. These are added depending on the first letter of the first name:

<table>
<tr><th>Chinese</th><th>English</th><th>Spanish</th></tr>
<tr><td>A-C (10x)</td><td>J-K (10x)</td><td></td></tr>
<tr><td colspan="2">D-I (10x)</td><td></td></tr>
<tr><td colspan="2"></td><td>N-O (3x)</td></tr>
</table>

This leaves us with 10 full time employees and 7 part time employees without any skills assigned.
The shifts that will require skills are the same for every day of the week.

Skills / Time	English	Chinese	Chinese, English	Spanish
0:00–8:00	2	2	2	0
8:00–16:00	1	1	1	1
16:00–0:00	2	2	2	0

During the second test creating new rosters will again result in no over- or understaffing. There are always enough employees that have the required skills to fill all the shifts.
In the next step the days off and preferences will be added.

Info / Employee	Day off	Prefer day off	Prefer day work	Prefer part of day off	Prefer part of day work
Al Baird					Evening
Alexis Rich					Evening
Daisy Fresh			Sunday		
Deb Smith				Night	
Jasmine Diaz		Monday			
Jeffrey Lee			Sunday		
Jennifer Han				Night	
Laura Grace		Monday			
Naomi Walls		Monday			
Nicole Holmes		Monday			
Oliver Curtis	Thursday				
Rachel Gold	Thursday				
Richard Stone	Thursday				

For a result we refer to the same roster as before in Appendix C.2. An optimal roster has been found and everyone has been given their preferences. There are no changes in the amount of kudos.

To test how kudos are recalculated on a larger scale we create a conflict. Oliver Curtis will now also have the preference of having Monday off. This way every employee that has the skill Spanish prefers Monday off. However one employee must be assigned this shift.
The entire roster can be found in Appendix C.2 and below are the kudos of the employees that have preferences set. Recall that employees without preferences don't get their kudos updated. $kudos_i$ is the amount of kudos for an employee after week i. $value_i$ is the value of the schedule given to an employee in week i. Between brackets the maximum value is shown.

		Week 1		Week 2		Week 3	
Info / Employee	$kudos_0$	$value_1$	$kudos_1$	$value_2$	$kudos_2$	$value_3$	$kudos_3$
Al Baird	50	10 (10)	47.83	10 (10)	45.83	10 (10)	44
Alexis Rich	50	10 (10)	47.83	10 (10)	45.83	10(10)	44
Daisy Fresh	50	6 (6)	47.83	6 (6)	45.83	6 (6)	44
Deb Smith	50	5 (5)	47.83	5 (5)	45.83	5 (5)	44
Jasmine Diaz	50	5 (5)	47.83	5 (5)	45.83	5 (5)	44
Jeffrey Lee	50	6 (6)	47.83	6 (6)	45.83	6 (6)	44
Jennifer Han	50	5 (5)	47.83	5 (5)	45.83	5 (5)	44
Laura Grace	50	5 (5)	47.83	5 (5)	45.83	5 (5)	44
Naomi Walls	50	2 (3)	71.74	3 (3)	68.75	3 (3)	66
Nicole Holmes	50	3 (3)	47.83	2 (3)	68.75	3 (3)	66
Oliver Curtis	50	3 (3)	47.83	3 (3)	45.83	2 (3)	66

It becomes clear from the results that after three weeks Naomi, Nicole and Oliver all got their preference twice and that their kudos is again equal.

The other employees see their kudos decrease and this may cause some discussion. One can argue that this is unfair as their preferences had nothing to do with the conflict of interest of Naomi, Nicole and Oliver. On the other hand all of those employees did get their preferences all the time where Naomi, Nicole and Oliver missed one of theirs. If in the future one of the other employees conflicts with either three this advantage will be saved.

If the current solution seems unfair and you wouldn't want the kudos of the uninvolved employees to be updated you could separate the employees in several groups. Although one should be careful doing this, as illustrated with the next example it will be hard to find these groups.

Example 6.3. We will look at the previous test in our example. Recall that the problem started with 3 people able to speak Spanish wanting a day off on Monday. To us it is clear that this will result in a problem because there must be one employee that speaks Spanish on Monday. Making a group of people with the exact same skills and preferences is most likely too precise and will single out everyone. Which defeats the purpose of comparing employees.
So making a group a bit larger, one might want to add everyone with the same preferences to one group. However in that case we would add Jasmine, who doesn't even speak Spanish and shouldn't be punished for this conflict. The same can be said for skills. Let's say an other employee speaks Spanish and prefers to work on Wednesdays. If this isn't a problem then it would again feel unfair to that employee if his kudos decrease.

7 Conclusions

7.1 Possible schedules

How useful it can be to create categories and possible schedules is really dependent on what the requirements for a roster are. Nursing rosters, for example, have only a relatively small number of different shifts that don't change throughout the weeks. This means that finding all the possible schedules won't have to be redone every week, except for those employees who changed their preferences and/or contract.
However if planning a roster requires a lot of different shifts, this method is not ideal. The time needed to find all possible schedules roughly doubles with each shift added. If the categories don't change this doesn't have to be a problem as they don't have to be recalculated. The number of schedules that are found also increases rapidly. At 10 shifts there are almost 200.000 different schedules saved which seems to be multiplied by 1.5 to 2 for each shift added.
For this reason this solution isn't feasible when the planning requires more than 10 shifts to be considered each day.

7.2 Kudos

Before looking at the four different parts we will look at the employees without preferences. To keep everything fair for these employees their kudos should either increase or stay the same. A decrease in kudos would not make sense if an employee didn't get anything he really wanted.
However there was a test in which the kudos of an employee without preferences decreases. For this reason we shouldn't update the kudos of employees without preferences. An increase afterwards can still be added if deemed fair, if so we should keep in mind that after doing this the average changed so scaling has to be applied again too.

7.2.1 Creating $c_{s,e}$

(a) $v_s + k_e$

(b) $v_s \cdot k_e$

(c) $\frac{v_s}{2s'} \cdot k_e$

(d) $(\frac{v_s}{2s'})^2 \cdot k_e$

(e) $\sqrt{\frac{v_s}{2s'}} \cdot k_e$.

As seen before using (a) doesn't do the job. The reason for this is that we can rewrite $\max \sum c_{s,e}x_{s,e} = \max \sum v_s x_{s,e} + \sum k_e x_{s,e}$. Because k_e is constant the maximum value found is independent of the kudos of the employees. Therefore the kudos will not be considered.
Method (b)-(e) give better results and all do consider the amount of kudos an employee has. For this reason we can conclude that we should multiply v_s and k_e in some way.
Other than that the methods gave the expected results. Comparing (b) to the others there is a difference in the contract hours. If a bad shift for a part time employee is considered equally bad as a bad shift for a full timer then (b) is the best way. But if this is considered worse the value should be divided by some factor depending on working time, which in our case was $2s'$. Another suggestion is the hours worked in that schedule.
Considering the latter three (c)-(e) the differences become visible when employees have several preferences. As expected using the linear (c) the first week was valued equally, but when the kudos weren't the same one employee was preferred because their slope is higher. Using the square (d) and square root (e) went as expected. When the value of a schedule was squared getting the first

bad shift is much worse then the next, which results in one employee getting the maximum and an other the maximum of what is left after that.
Using the square root of the value results in spreading the bad shifts over different employees. In my personal opinion this would be more fair, but both methods could be used depending on what you want to achieve.
Note that this isn't even close to everything that can be tested. For example combinations of the tested method can be used such as $v_s^2 \cdot k_e$, using the square without considering shifts. Or higher powers or roots. These results do however give some handholds on what should be evaded or considered when trying to achieve a certain way to distribute preferred shifts over employees. Also keep in mind that (strictly) increasing functions should always be considered.

7.2.2 Recalculating kudos

(a) $k_e - v_s + s'$

(b) $k_e \cdot \frac{s'}{v_s}$

(c) $\begin{cases} k_e \cdot \frac{s'}{v_s} & v_s \geq s' \\ k_e \cdot (-\frac{v_s}{s'} + 2) & v_s < s'. \end{cases}$

(d) $k_e \cdot \frac{m_s}{v_s}$

When considering differences in value of part and full time employees a big difference in (a) and (b)-(d) was noticeable. Because the ratio between the values depends on the number of shifts s', and the amount of kudos added using (a) does not depend on this ratio, it may take an unexpected large amount of time to get to the given point where the full time employee finally is given the advantage. This doesn't immediately mean that (a) is bad and (b)-(d) are good. In fact from this we can conclude that it is important to note by how much you increase the kudos. Making the amount added larger could solve the problem found when using (a). This difference is illustrated best in 6.2.7.

Although (b) and (c) didn't seem to be very different, they even are partially the same, there is something interesting happening here. Looking at 6.2.7 again, a slight difference can be noticed. The reason for this can be explained by the amount of kudos added/removed in one week. Which method is better depends on the goals and what is deemed fair.
Finally there was a fourth method, (d), added to the tests. This method uses the potential maximum value considering the preferences instead of the theoretical maximum of $2s'$. This seems like the best way to go otherwise an optimal schedule might be of value s' and will be considered half as good as it should be. This will result in a larger increase for this employee compared to other employees, which may be considered unfair. An other method could be to add value to a schedule when not preferred shifts aren't assigned.

7.2.3 Scaling kudos

During the testing it was preferred to use a multiplicative way of scaling. However this wasn't necessarily because using an additive way was bad. As seen in 6.2.5 and 6.2.6 the combinations of updating and scaling is important. Using additive updating and multiplicative scaling doesn't go well together and the same goes for multiplicative updating and additive scaling.

7.2.4 Average kudos

A difference in average was tested only briefly. The idea behind using a higher or lower average was that shifts that are just as good as others are bad, might have some other ratio when using a lower or higher average. However no significant changes were noticed.
It is therefore recommended to keep the average at 50. This way an employee won't get 0 or 100 kudos very fast at which point their kudos can't decrease or increase anymore. It also seems more natural to keep the average right between the minimum and maximum.

7.3 Model

Looking at the validity of the model we can be content. As can be verified in Appendices C.1 and C.2, there is no over- or understaffing, so the rosters found are optimal. Also, all shifts with skills are assigned correctly and where possible the preferences are given correctly. When this wasn't possible we saw, by updating the kudos, that every employee is treated as equal as possible. We did not consider the speed of the model too much because this wasn't the focus for the thesis. For the examples tested speed wasn't an issue, but it should be tested more thoroughly if the model is used for a larger number of employees.

8 Future research

The way kudos are used throughout the thesis is just the start of what can be done. A lot of factors can be adjusted to achieve the goal that you would want to achieve. We can, for example, adjust the way a value v_s is calculated or how $c_{s,e}$ can be determined depending on v_s and k_e. In short we will give some suggestions that can be researched to improve the results.

One might not want to create all possible schedules for two related reasons: the time it takes and the number of schedules can be large. A workaround could be to make a small selection of schedules and, if certain schedules seem better than others, choose to make small adjustments to them to find the best schedules without having to save all the possible schedules.

The value v_s now depends on the number of shifts and the number of preferences in a schedule. This can be adjusted so that certain preferences can be given a weight in the case that one preference is deemed more important than an other. This can be useful if the reason for a preference is known.
Some adjustments can be made for the comparison between one good and bad shift and no preferences. We considered this as equal, but this can be adjusted to be worth more or less, depending on the goal.
An other change that can be researched is to increase the value when someone doesn't work a shift that they didn't prefer, instead of not decreasing the value, which currently is being done.
Many other factors can increase the value of a schedule apart from their preferences. For example the number of night shifts, the length between two shifts, the length of a weekend, the days on which the weekend occurs etc. All of these factors can be used to assign a value to a schedule. In some cases these might be personal, in which case they can be added when determining $c_{s,e}$ instead of with v_s.

For $c_{s,e}$ the options are endless. The results provide some indication on which method to use depending on what one may find important. Picking one of the options in the thesis gives a good starting point and adjustments can then be made. To make the results more extreme one could, for example, use v_s^3 instead of using the square. Or it can be made less extreme by using $v_s^{3/2}$.
Considering the examples above, one factor might be considered more important than others. Addressing these options with different factors can also be researched. E.g, the length of a weekend is important and will be squared, but the break between shifts is less important and will be contributing linearly.
Right now the kudos k_e are always considered linear when determining $c_{s,e}$. It can be interesting to test what happens when using the square or square root of k_e when calculating $c_{s,e}$.

Over- and understaffing is currently avoided completely. The cost O can be changed to a lower number if giving preferences has a higher priority. The same goes for the value $Z_{g,g'}$.
If an employee has multiple skills one can also add the possibility to prefer shifts that require a certain skill that an employee likes.

References

[1] Abdullah Alsheddy and Edward Tsang, *Empowerment scheduling for a field workforce*, Journal of Scheduling, **14**, 2011, pp. 639–654.

[2] B. Teahan. *Implementation of a self-scheduling system: a solution to more than just schedules*, Journal of Nursing management, **6**, 1998, pp. 361–368.

[3] S.I. Gass and C.M. Harris, *Strong duality theorem*, Encyclopedia of Operations Research and Management Science, 2001, pp. 790–790

[4] Richard Karp, *Reducibility among combinatorial problems*, Proceedings of a Symposium on the Complexity of Computer Computations, 1972, pp. 85–103.

[5] Alexander Schrijver, *Theory of linear and integer programming*, 1986.

[6] Irv Lustig, *Interview with George Dantzig*, `https://www.informs.org/Explore/History-of-O.R.-Excellence/Oral-Histories/George-Dantzig`.

[7] M. Todd and E. Yildirim, *On Khachiyan's algorithm for the computation of minimum-volume enclosing ellipsoids*, Discrete Applied Mathematics, **155**, 2007, pp. 1731–1744.

[8] Jorne van den Bergh et al, *Personnel scheduling: A literature review*, European Journal of Operational Research, **226**, 2013, pp. 367–385.

[9] Sandjai Bhulai, Ger Koole and Auke Pot, *Simple Methods for Shift Scheduling in Multiskill Call Centers*, Manufacturing & Service Operations Management, **10**, 2008, pp. 411 – 420.

[10] *Types of optimization solvers*, `https://www.ibm.com/analytics/optimization-solver`.

Appendices

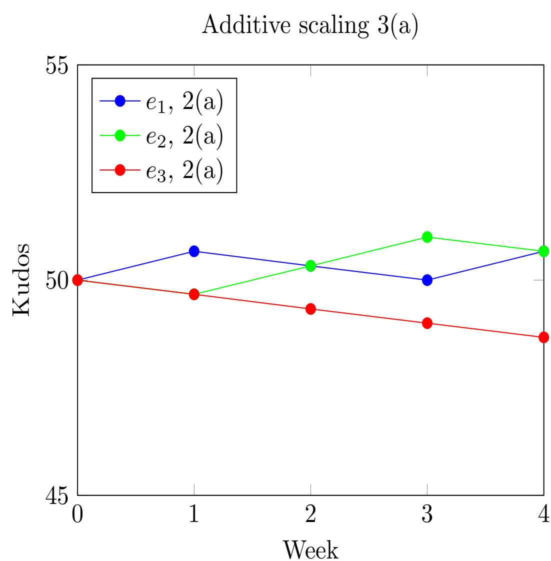
A Possible schedules results

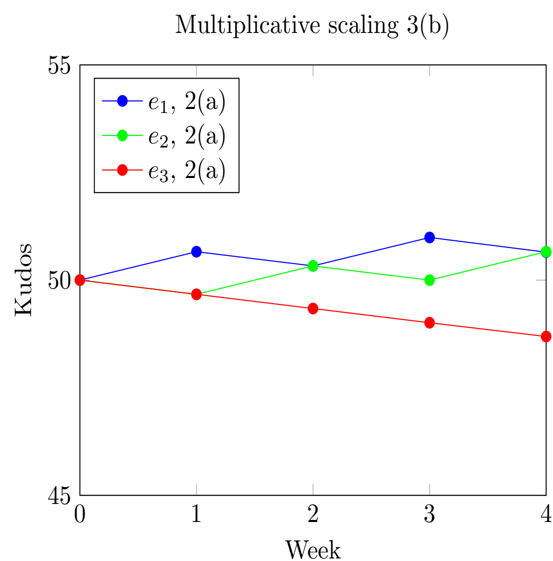
The following letters indicate properties of the categories: (a) contract hours, (b) Thursday off, (c) holiday of Friday, (d) prefers Wednesday off and (e) prefers mornings off. Followed by the number of schedules depending on the number of shifts on a day.

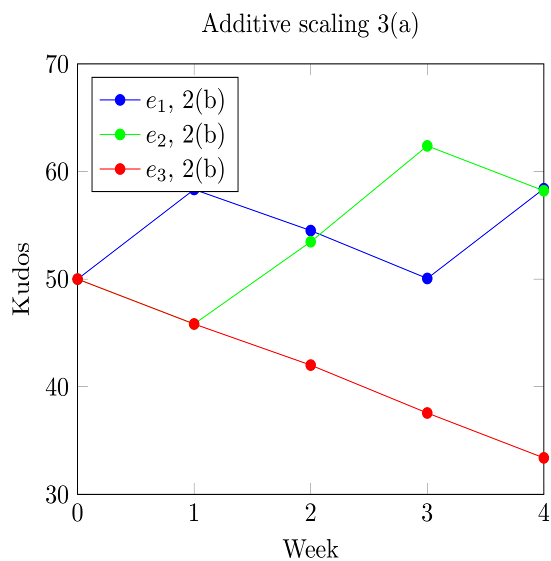
(a)	(b)	(c)	(d)	(e)	1	2	3	4	5	6	7	8	9	10
20h	red	red	red	red	55	290	824	2013	3844	6260	9908	15087	21268	28786
20h	red	red	red	green	55	290	824	1979	3810	6226	9636	14169	20350	27868
20h	red	red	green	green	55	290	824	1939	3760	6166	9396	13614	19705	27133
20h	red	green	red	red	40	188	512	1210	2278	3688	5796	8766	12318	16636
20h	red	green	red	green	40	188	512	1191	2259	3669	5644	8253	11805	16123
20h	red	green	green	green	40	188	512	1165	2227	3631	5492	7899	11397	15661
20h	green	red	red	red	35	184	521	1250	2373	3868	6097	9230	12993	17576
20h	green	red	red	green	35	184	521	1230	2353	3848	5937	8690	12453	17036
20h	green	red	green	green	35	184	521	1200	2313	3798	5737	8240	11913	16406
20h	green	green	red	red	25	109	286	665	1238	1984	3102	4680	6554	8824
20h	green	green	red	green	25	109	286	655	1228	1974	3022	4410	6284	8554
20h	green	green	green	green	25	109	286	637	1204	1944	2902	4140	5960	8176
38h	red	red	red	red	6	56	252	1960	6174	12144	28408	67998	120552	195456
38h	red	red	red	green	6	56	252	1572	5640	11468	22136	41070	87522	156216
38h	red	red	green	green	6	56	252	1325	5121	10666	18928	32899	73593	135566
38h	red	green	red	red	6	49	201	1228	3627	7022	15401	34192	59668	95384
38h	red	green	red	green	6	49	201	1047	3381	6712	12593	22582	45439	78464
38h	red	green	green	green	6	49	201	866	2991	6084	10087	16456	34865	62413
38h	green	red	red	red	2	24	120	792	2508	5200	11952	27146	48608	79952
38h	green	red	red	green	2	24	120	654	2310	4940	9568	17216	36104	64712
38h	green	red	green	green	2	24	120	566	2109	4596	8192	13926	30335	55744
38h	green	green	red	red	2	21	96	540	1615	3276	7272	15974	28148	45752
38h	green	green	red	green	2	21	96	460	1503	3131	5952	10532	21365	37547
38h	green	green	green	green	2	21	96	372	1302	2787	4576	7242	15596	28579

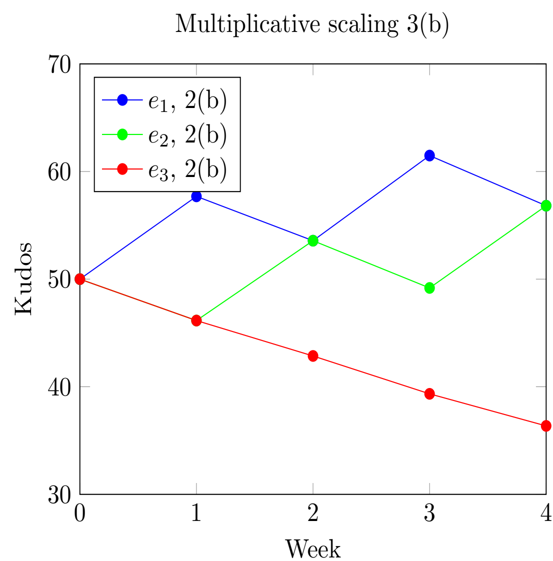
B Kudos results

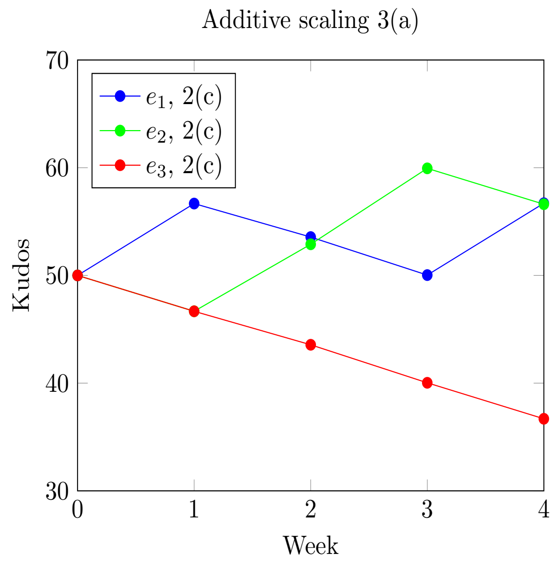
B.1 Prefer to have a day off on the same day

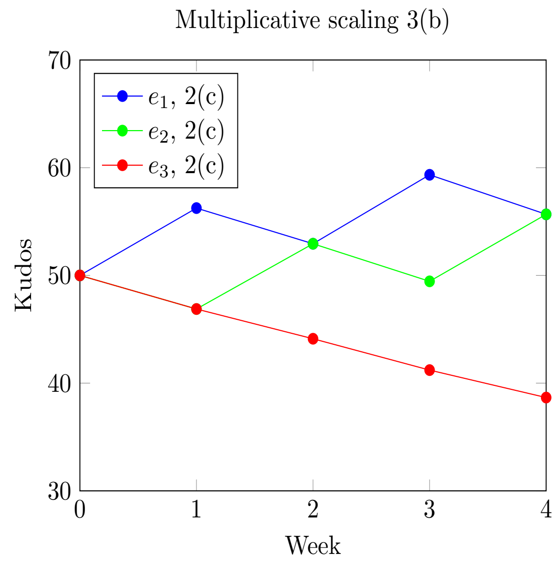




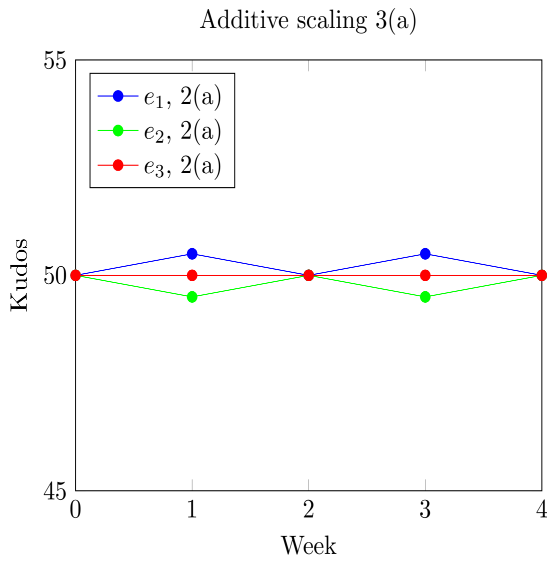


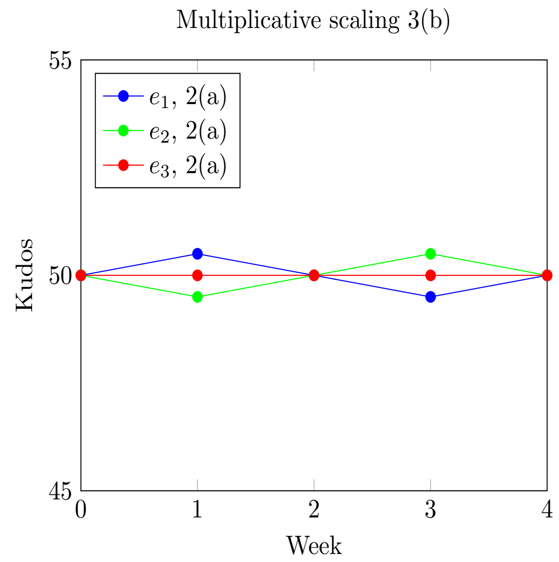


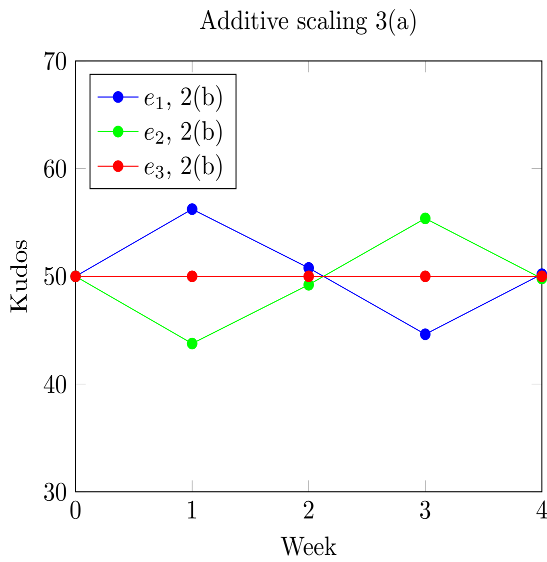

Additive scaling 3(a)
70
60
50
40
30
Kudos
0
1
2
3
4
Week
e_1, 2(c)
e_2, 2(c)
e_3, 2(c)

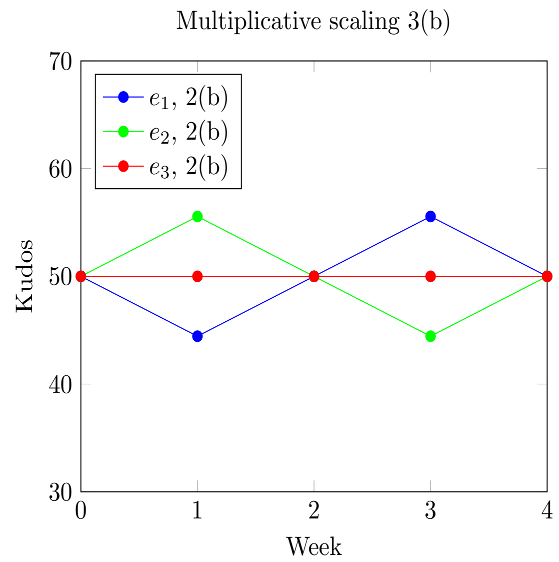

Multiplicative scaling 3(b)
70
60
50
40
30
Kudos
0
1
2
3
4
Week
e_1, 2(c)
e_2, 2(c)
e_3, 2(c)

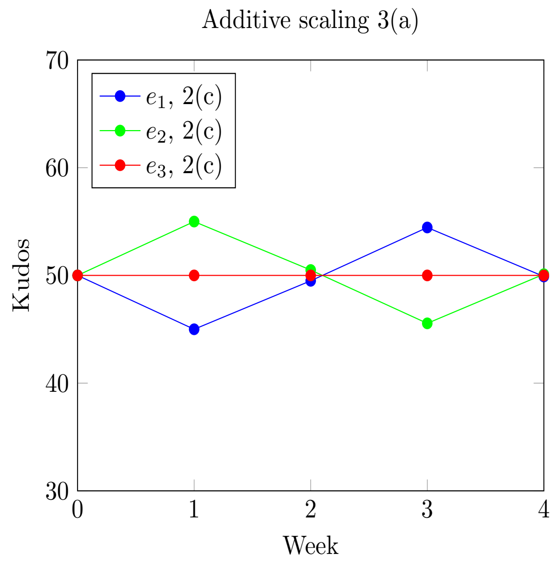
B.2 Prefer the same day off revisited

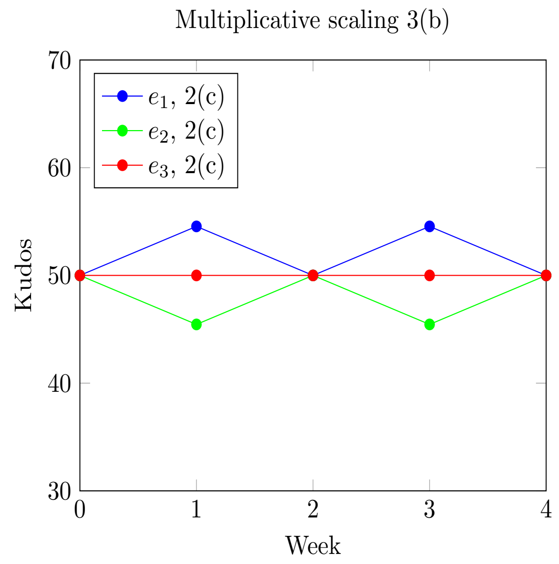




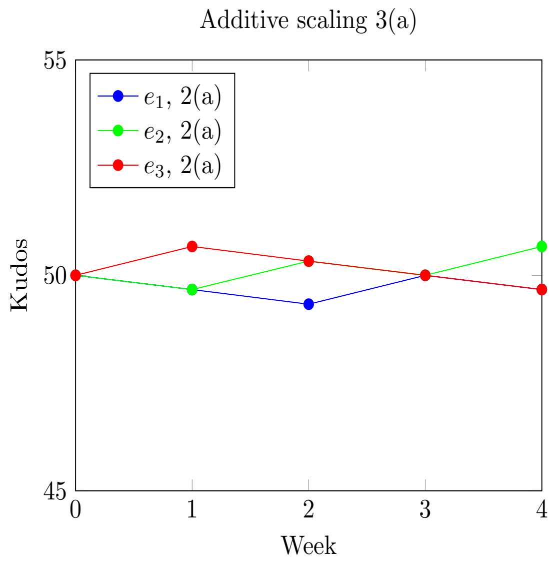


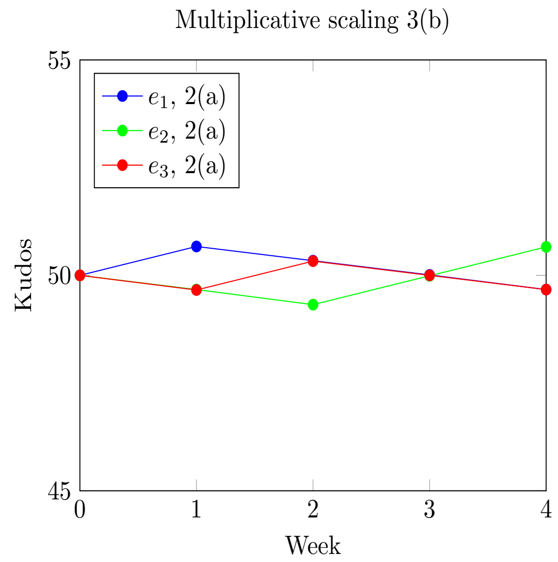


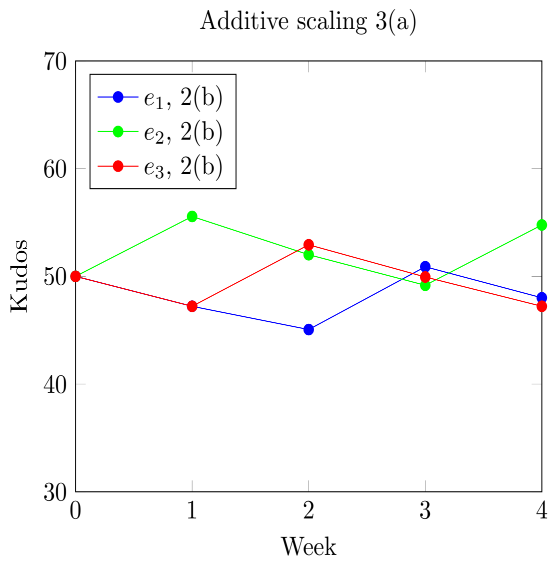

Additive scaling 3(a)
70
60
50
40
30
Kudos
0
1
2
3
4
Week
e_1, 2(c)
e_2, 2(c)
e_3, 2(c)

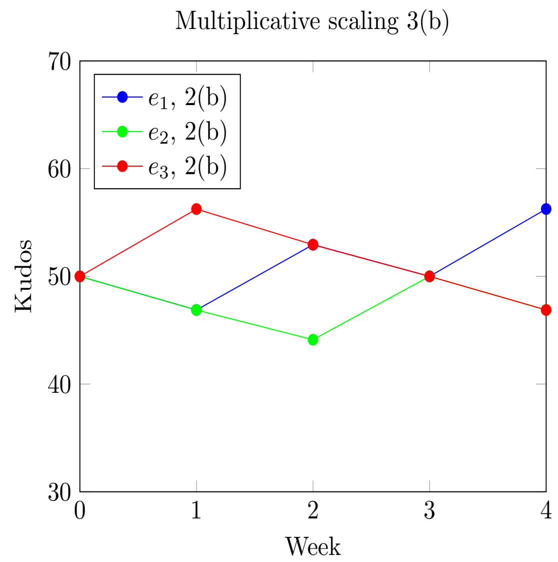

Multiplicative scaling 3(b)
70
60
50
40
30
Kudos
0
1
2
3
4
Week
e_1, 2(c)
e_2, 2(c)
e_3, 2(c)

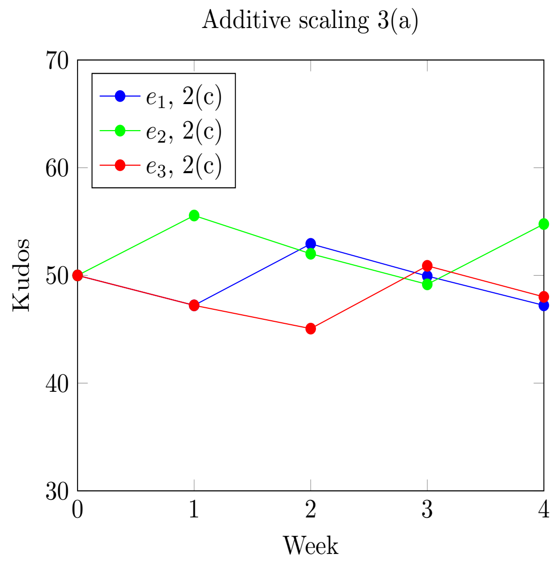
B.3 Conflict with three employees

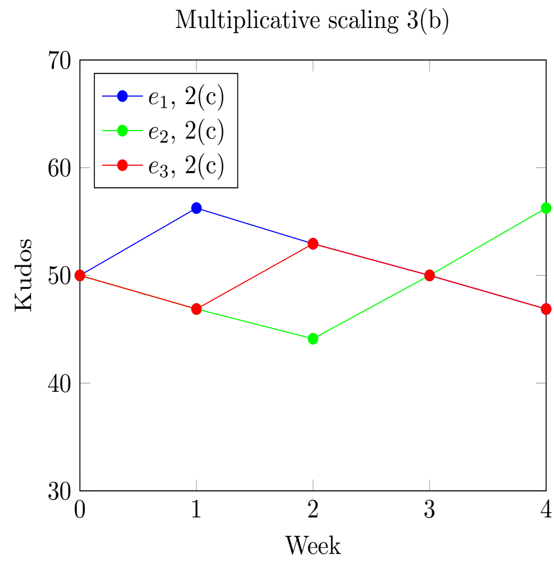




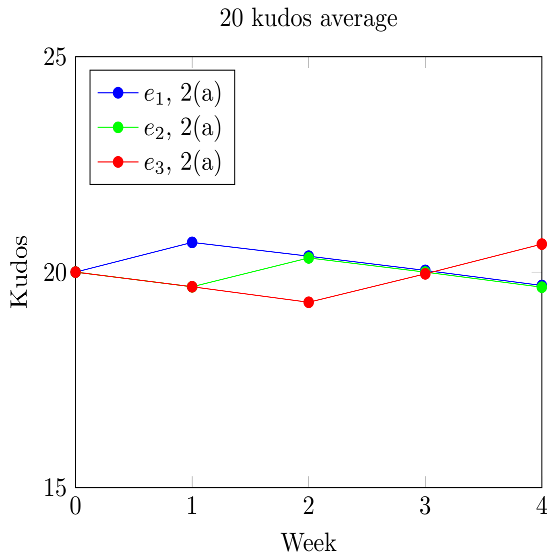


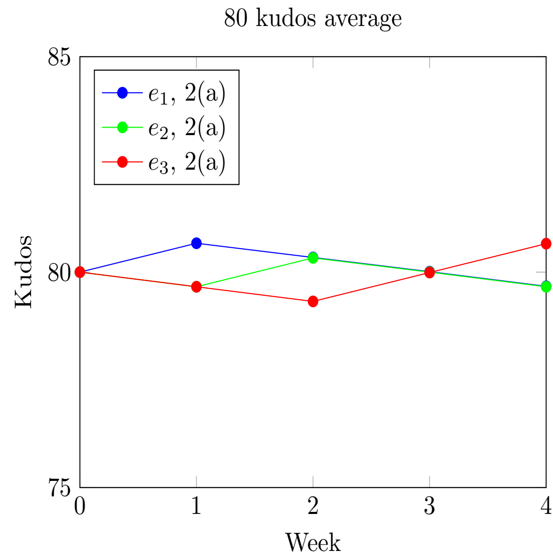


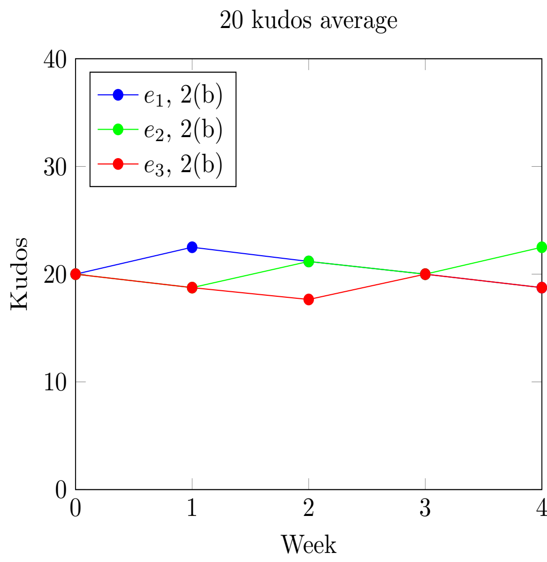

Additive scaling 3(a)
e_1, 2(c)
e_2, 2(c)
e_3, 2(c)
Kudos
Week
70
60
50
40
30
0
1
2
3
4

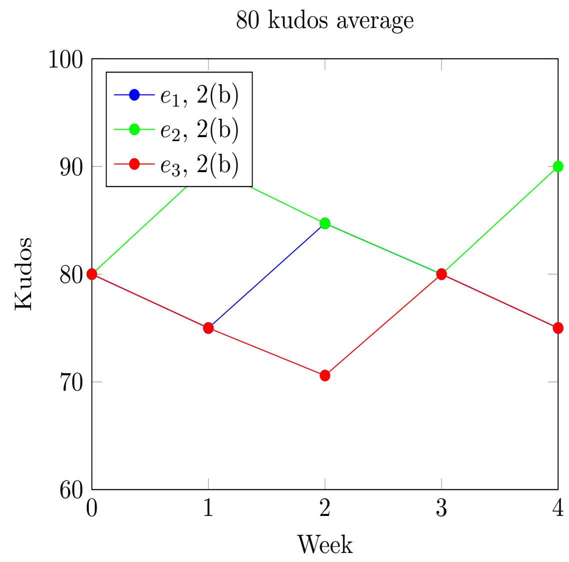

Multiplicative scaling 3(b)
e_1, 2(c)
e_2, 2(c)
e_3, 2(c)
Kudos
Week
70
60
50
40
30
0
1
2
3
4

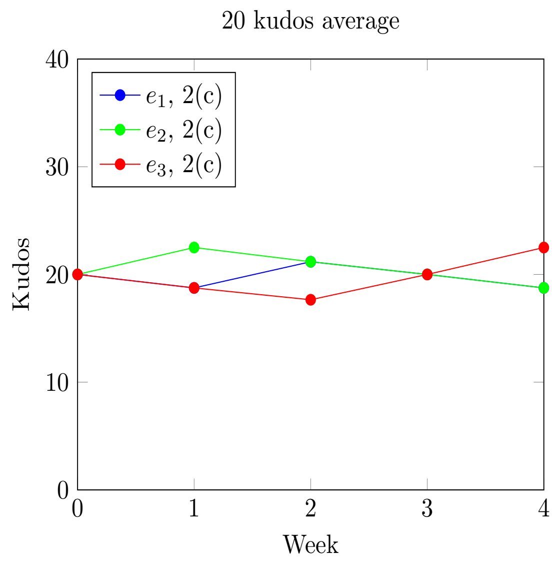
Average kudos:



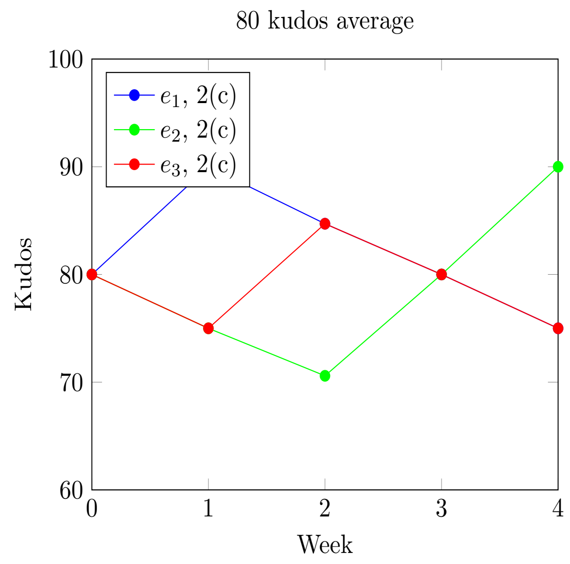






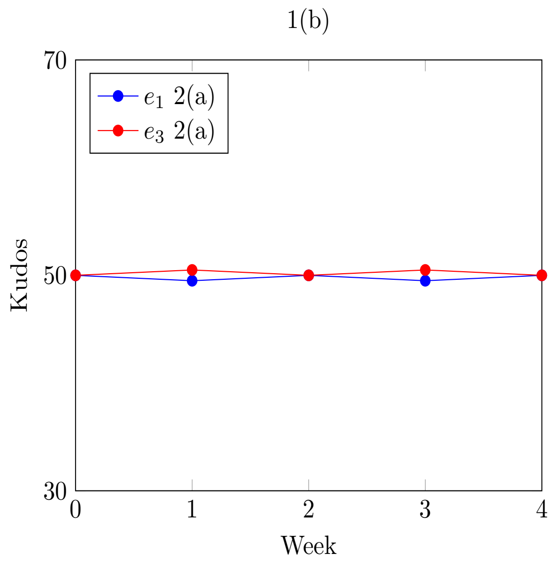


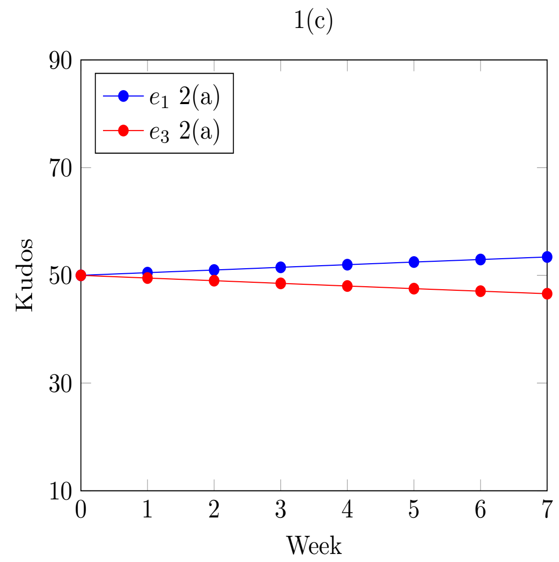
20 kudos average
40
30
20
10
0
Kudos
0
1
2
3
4
Week
e_1, 2(c)
e_2, 2(c)
e_3, 2(c)

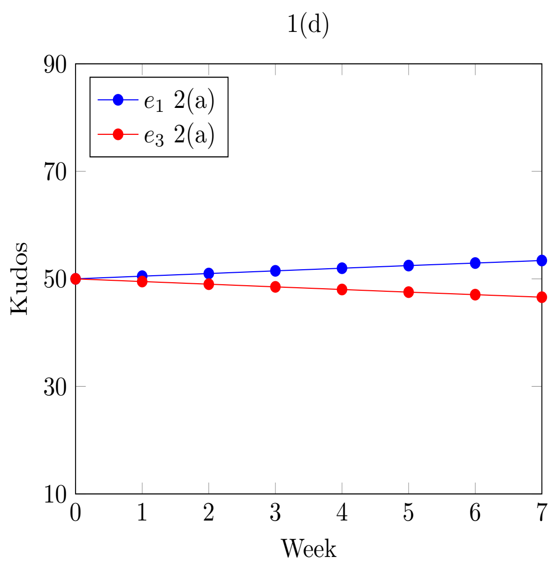


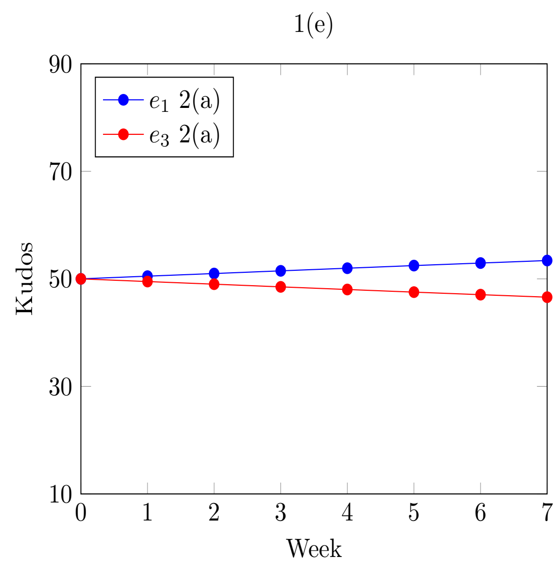
80 kudos average
100
90
80
70
60
Kudos
0
1
2
3
4
Week
e_1, 2(c)
e_2, 2(c)
e_3, 2(c)

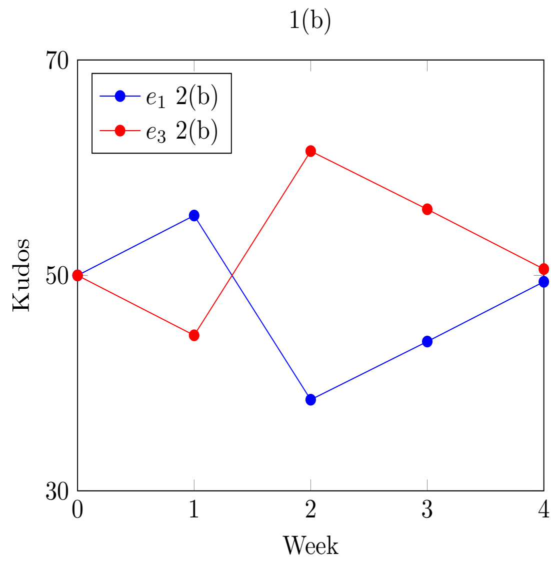
B.4 Behaviour with different contract hours

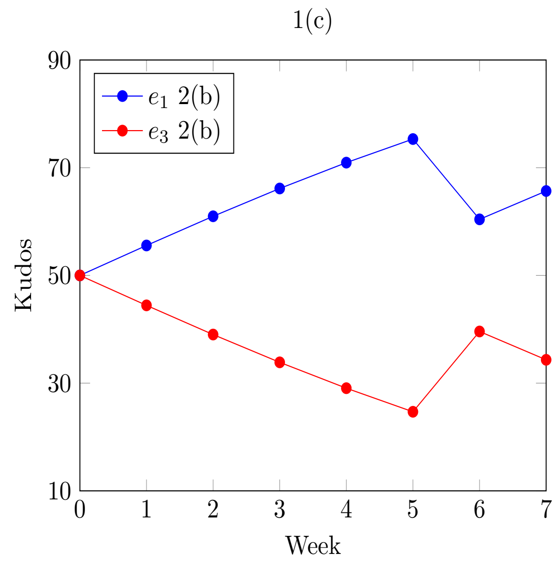


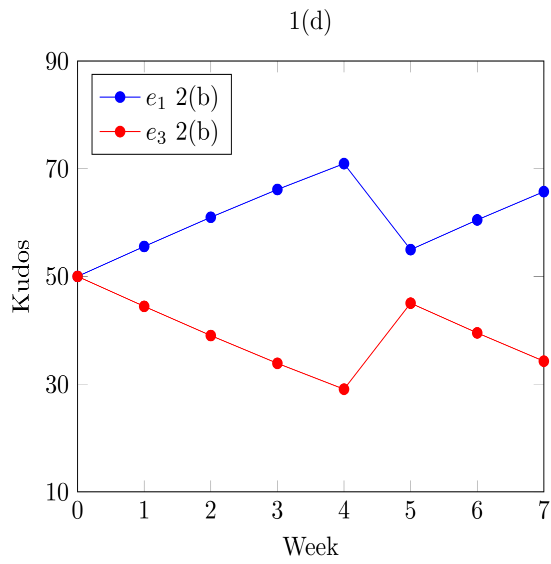


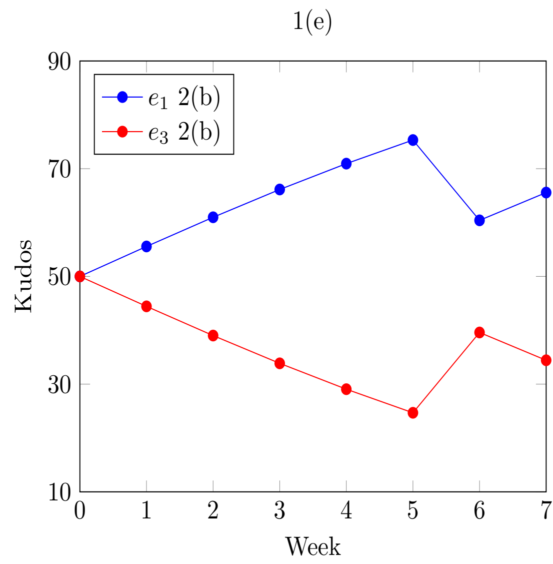


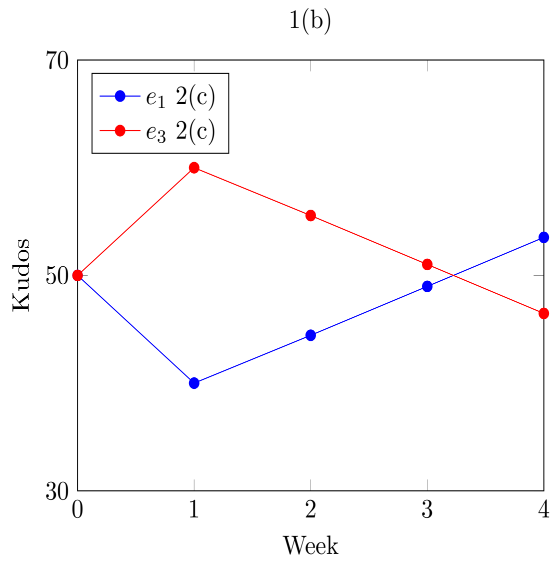


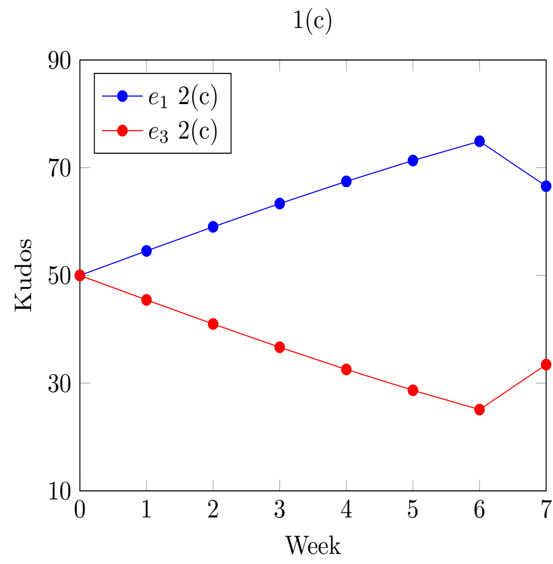

1(b)
e_1 2(b)
e_3 2(b)
Kudos
Week
70
50
30
0
1
2
3
4

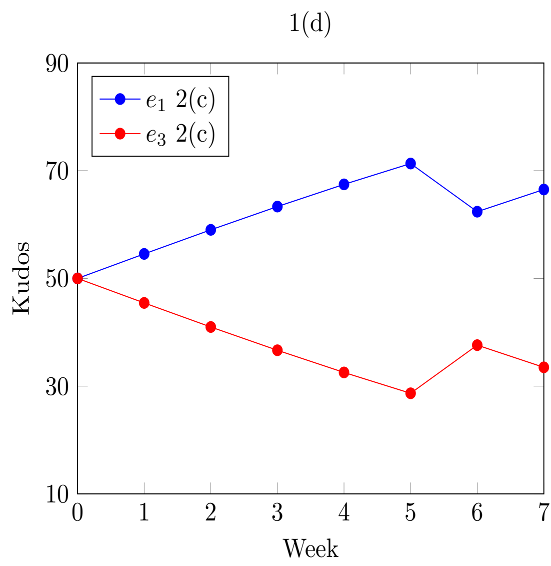

1(c)
e_1 2(b)
e_3 2(b)
Kudos
Week
90
70
50
30
10
0
1
2
3
4
5
6
7

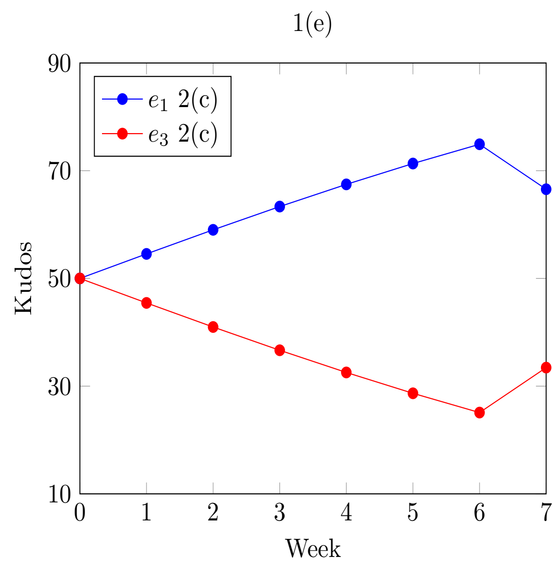

1(d)
e_1 2(b)
e_3 2(b)
Kudos
Week
90
70
50
30
10
0
1
2
3
4
5
6
7


1(e)
e_1 2(b)
e_3 2(b)
Kudos
Week
90
70
50
30
10
0
1
2
3
4
5
6
7


1(b)
70
50
30
Kudos
0
1
2
3
4
Week
e_1 2(c)
e_3 2(c)


1(c)
90
70
50
30
10
Kudos
0
1
2
3
4
5
6
7
Week
e_1 2(c)
e_3 2(c)


1(d)
90
70
50
30
10
Kudos
0
1
2
3
4
5
6
7
Week
e_1 2(c)
e_3 2(c)


1(e)
90
70
50
30
10
Kudos
0
1
2
3
4
5
6
7
Week
e_1 2(c)
e_3 2(c)

C Model results

C.1 Test with 16 employees

No skills or preferences and the required shifts number of employees are as shown.

Time \ Day	Monday	Tuesday	Wednesday	Thursday	Friday	Saturday	Sunday
6:00–14:00	5	6	6	5	5	3	2
14:00–22:00	4	5	5	4	4	3	3

	Monday		Tuesday		Wednesday		Thursday		Friday		Saturday		Sunday	
	6:00	14:00	6:00	14:00	6:00	14:00	6:00	14:00	6:00	14:00	6:00	14:00	6:00	14:00
Al Baird	■		■			■					■			■
Deb Smith	■		■		■		■						■	
Isaac Haynes	■		■		■			■		■				
Jess Lester		■					■		■		■			■
Leslie Woods		■					■		■		■			■
Lorn Olson	■		■		■			■		■				
Lorraine Jennings		■		■		■		■		■				
Meredith Miles		■		■		■		■		■				
Naomi Walls					■		■							
Nicole Holmes			■		■							■		
Oliver Curtis			■		■							■		
Rachel Gold				■					■					
Richard Stone				■		■						■		
Ricky Rookie						■			■					
Rita Sanders	■						■							
Rob Lewis				■					■				■	

For the second roster skills are involved. Every first shift of the day requires one employee with the skill Chinese and the second shift requires English. The following employees have these skills.

English	Chinese
Isaac Haynes	Jess Lester
Meredith Miles	Meredith Miles
Rita Sanders	Rachel Gold
■	Rob Lewis

	Monday		Tuesday		Wednesday		Thursday		Friday		Saturday		Sunday	
	6:00	14:00	6:00	14:00	6:00	14:00	6:00	14:00	6:00	14:00	6:00	14:00	6:00	14:00
Al Baird		■		■		■					■		■	
Deb Smith		■		■		■						■		■
Isaac Haynes		■		■		■		■		■				
Jess Lester	■		■			■		■					■	
Leslie Woods	■		■		■		■		■					
Lorn Olson	■		■		■			■		■				
Lorraine Jennings	■		■		■			■		■				
Meredith Miles	■		■		■		■		■					
Naomi Walls			■						■					
Nicole Holmes		■		■								■		
Oliver Curtis					■		■							■
Rachel Gold							■				■			
Richard Stone				■		■				■				
Ricky Rookie					■		■		■					
Rita Sanders												■		■
Rob Lewis									■		■			

The third part contains restrictions and preferences. These are:
Days off: Richard Stone and Ricky Rookie on Wednesday, Nicole Holmes and Rita Sanders on Friday, Oliver Curtis on Saturday and Sunday.
Preferred days off: Rachel Gold and Richard Stone on Tuesday.
Preferred working part of day: Al Baird and Leslie Woods in the morning, Lorraine Jennings and Meredith Miles in the evening.
The roster found is given below.

	Monday		Tuesday		Wednesday		Thursday		Friday		Saturday		Sunday	
	6:00	14:00	6:00	14:00	6:00	14:00	6:00	14:00	6:00	14:00	6:00	14:00	6:00	14:00
Al Baird	■		■		■		■		■					
Deb Smith		■		■		■						■		■
Isaac Haynes	■		■		■			■		■				
Jess Lester	■		■						■		■		■	
Leslie Woods	■		■		■		■		■					
Lorn Olson	■		■		■		■			■				
Lorraine Jennings						■		■		■		■		■
Meredith Miles		■		■						■		■		■
Naomi Walls			■		■			■						
Nicole Holmes				■		■								
Oliver Curtis				■		■		■						
Rachel Gold					■		■							
Richard Stone		■							■		■			
Ricky Rookie							■		■				■	
Rita Sanders				■		■								
Rob Lewis		■									■			

C.2 Test with 48 employees

	Monday			Tuesday			Wednesday			Thursday			Friday			Saturday			Sunday		
	0:00	8:00	16:00	0:00	8:00	16:00	0:00	8:00	16:00	0:00	8:00	16:00	0:00	8:00	16:00	0:00	8:00	16:00	0:00	8:00	16:00
Al Baird			■			■			■			■								■	
Alexis Rich							■			■			■				■				■
Ana Levy		■				■			■							■				■	
Ashley Preston	■			■			■			■			■								
Beau Harris			■			■							■			■			■		
Bill Jean							■			■			■			■			■		
Cathy Johnson			■			■			■			■							■		
Chad Sanford							■			■			■				■				■
Christine Westin							■			■			■					■			■
Craig Parker		■			■			■			■				■						
Daisy Fresh		■			■				■									■			■
Deb Smith			■			■								■				■			■
Dona Krieg	■			■			■			■				■							
Elaine Stuart								■			■			■			■			■	
Emily Warner								■			■			■				■			■
Emma Buckley			■			■								■			■			■	
Frank Junior	■			■			■				■			■							
Haley Wallace	■				■			■								■			■		
Isaac Haynes			■			■			■								■				■
Ivone Dawn		■			■				■			■								■	
Jasmine Diaz							■			■			■			■			■		
Jeffrey Lee			■			■			■			■			■						
Jennifer Han	■			■				■								■			■		
Jeremy Murphy		■			■			■				■			■						
Jess Lester								■			■			■			■			■	
Jim Carrey							■			■				■			■			■	
John Locke			■			■			■			■			■						
Julian Shepherd		■				■			■							■			■		
Keith Spencer		■			■				■								■			■	
Kim Basinger		■			■			■									■			■	
Laura Grace	■			■				■				■			■						
Laurel Ward	■			■				■			■			■							
Leslie Woods		■			■			■										■			■
Liz Luther		■			■			■				■							■		
Lorn Olson			■			■			■			■			■						
Lorraine Jennings		■				■			■			■			■						
Marissa Stokes			■			■								■				■			■
Maureen Carter	■			■			■				■			■							
Meredith Miles		■			■				■									■			■
Michele Hurley								■			■			■			■			■	
Naomi Walls								■			■						■				
Nicole Holmes					■								■								
Oliver Curtis					■										■			■			
Rachel Gold				■							■										
Richard Stone					■										■			■			
Ricky Rookie				■							■										
Rita Sanders					■										■			■			
Rob Lewis				■							■										