Master Thesis
ICA-3230570

# Nurse rostering through linear programming and repair heuristics 

Author:<br>Tim van Weelden<br>timvanweelden@gmail.com

Supervisor:
Dr. Han Hoogeveen
J.A.Hoogeveen@uu.nl

Secondary Supervisor:
Dr. Ir. Marjan VAN DEN AKKER
J.M.vandenAkker@uu.nl

Department of Information and Computing Sciences,
Faculty of Science
Utrecht University, The Netherlands


#### Abstract

We consider a nurse scheduling problem in a large hospital in the center of The Netherlands. Approximately 50 nurses with different qualifications should obtain a work schedule for a period of 6 weeks. Every day is divided in three shifts (day, late and night) and we should make sure enough employees with the correct qualifications get assigned to each shift. While creating a solution, we should both take the general regulations and the personal roster preferences into account.

We present a three-stage solution approach. First, we create a set of suitable schedules for every employee using an individual roster generation scheme. Thereafter, all individual schedules are combined into a full schedule which satisfies all occupancy demands for the period. A Linear Program is used for this step. Finally, some remaining problems are resolved using a set of proposed repair heuristics. These heuristics can also be used to perform some schedule changes afterwards.

Our results look promising, although it is difficult to cope with all possible preferences and demands.


## Contents

Preface ..... 4
1 Introduction and main goal ..... 5
1.1 Employee qualifications ..... 6
1.2 Appointment contracts ..... 7
1.3 Forwards rotating rosters ..... 8
1.4 Default hard restrictions for schedules ..... 10
1.5 Default soft restrictions for schedules ..... 12
1.6 Minimal occupancy demands ..... 13
1.7 Data analysis ..... 14
1.8 Main goal and assumptions ..... 17
1.9 Thesis structure ..... 17
2 Algorithmic analysis ..... 18
2.1 Local search ..... 19
2.2 Linear (and constraint) programming ..... 22
2.3 A basic approach by Hoogeveen and Penninkx ..... 28
3 The main approach ..... 32
3.1 Legend of constants and variables ..... 32
3.2 Master problem with task assignment ..... 34
3.3 Introducing qualification groups ..... 35
3.4 The UMC master problem ..... 36
3.5 Solving the ILP ..... 39
4 Creation of individual rosters ..... 41
4.1 Depth-first search with forwards rotation ..... 41
4.2 Rostering for 6 weeks: a mathematical challenge ..... 47
4.3 Soft requirements and costs of individual rosters ..... 52
4.4 Special generators for problematic employees ..... 54
5 ILP results and improvements ..... 56
5.1 Adding roster mutations before solving the ILP ..... 57
5.2 Some extensions of the method ..... 59
5.3 Results for February - March 2013 ..... 63
6 Heuristics to repair imperfections ..... 66
6.1 Introduction and definitions ..... 67
6.2 An order for repairing rosters ..... 73
6.3 Auxiliary functions for repairing ..... 75
6.4 Allocation of a hard repair requirement ..... 79
6.5 Resolving night shift shortages ..... 82
6.6 Resolving day and late shift shortages ..... 86
6.7 Possible extensions ..... 87
6.8 Experiments and results ..... 91
7 Conclusion and discussion ..... 95
7.1 Conclusion ..... 95
7.2 Discussion ..... 97
7.3 Future research ..... 100
A Extraction of the CBA ..... 104
B List of duty codes ..... 109
C Employee preferences \& demands ..... 110
D Example roster ..... 114
E Experiments on mutation ..... 117
E. 1 Setup of the experiments ..... 117
E. 2 Overview of the results ..... 118
E. 3 Additional test on roster period February - March 2013 ..... 118
F Results of the ILP phase ..... 120
F. 1 Input data ..... 120
F. 2 Parameter settings ..... 129
F. 3 The roster and demand satisfaction ..... 131
G Results of the repair phase ..... 138
G. 1 Basic repairing ..... 138
G. 2 Advanced repairing ..... 143
"Een mens lijdt dikwijls 't meest Door 't lijden dat hij vreest Doch dat nooit op komt dagen. Zo heeft hij meer te dragen Dan God te dragen geeft."

Possible author:
Jacobus Revius (1586-1658)[11, 23]

## Preface

This document is the master thesis of Tim van Weelden, student Computing Sciences at Utrecht University, The Netherlands. It is the final part of my thesis project in which we have launched a study to explore methods for nurse rostering in a $24 / 7$-environment. The project has been supervised by Han Hoogeveen and Marjan van den Akker and has been executed in association with Floris van der Laan of the University Medical Center (UMC) Utrecht.

I would like to thank Han Hoogeveen and Marjan van den Akker for their supervising support. Second, I want to thank Floris van der Laan of the UMC Utrecht for introducing the case and for his time investment.

I am very grateful to a number of other people for having some stimulating conversations or helping to gain inspiration. These include my family and friends, like Dennis, Ewoud, Michelle, Mascha, Wim and Paula, who were always there for me when I needed them.

I certainly would like to thank Marijke for firstly pre-reading a large part of the thesis report and secondly being a great support during the last months of the project.
Last, but certainly not least, I would like to thank my mother, whose support meant a lot to me.

Tim van Weelden
May 6, 2013

## Chapter 1

## Introduction and main goal

The department of cardiothoracic surgery is one of the many departments of the University Medical Center (UMC) Utrecht, one of the biggest hospitals in the center of The Netherlands. The department mainly treats patients who suffer from heart diseases and pulmonary affection and provides several treatment rooms (at the same time dormitories) at different care levels.

Taking care of the patients (washing, providing food and drugs) in this unit is mainly done by the approximately 52 department's nursing employees. Most of them have a contract for a permanent appointment size, but there are also a few with a more flexible contract. Apart from them, the department's workforce also consists of the unit direction and some administrative employees, whom we do not consider in this thesis problem.

The department is a $24 / 7$ environment, which means that at any time of the day, a collection of nurses with the appropriate qualifications should be available to take care. The daily required number of nurses is spread over three shifts of approximately 8 hours each: an early (day) shift ( $07: 30 \mathrm{~h}-16: 15 \mathrm{~h}$ ), a late shift ( $15: 15 \mathrm{~h}-23: 45 \mathrm{~h}$ ) and a night shift $(23: 15 \mathrm{~h}-07: 45 \mathrm{~h})$. There is an overlap between the shifts, to provide time for work transfer. The occupancy demands (for several skill categories which a nurse can have) for a shift differ among the various shifts on a day.

The employees each have there own preferences regarding their schedule. They can prefer to have a particular (part of a) day off, a specific shift on a certain day (in each week, or incidentally), or they can be indifferent. Furthermore, some employees provide restrictions on the number of times they have a particular shift during the week, for instance because they do not like a particular shift, or they want to spend the evenings with their family. Night shifts provide a special kind of restriction: some people hate them and others love them. Basically, every employee has to perform them from time to time, but the scheduler tries to assign them as little as possile to employees who dislike them.

Finally, some employees should perform special duties from time to time, which varies from study days, to incorporation of students, to special 'quality' days.

Rosters are created for a period of $k$ weeks. At this moment, $k=6$, meaning that a roster determines all the individual employee schedules for a period of six weeks. It was initially said that it was allowed to change $k$ into a different number. Preferably, $k$ should be equal to at least 4 , since according to the hospital regulations, employees should know their roster a few weeks in advance and it is not prefered to have the hassle of the roster designment process too often. Furthermore, when an employee must know his incidental preferences for days off during the schedule to be designed (like birthdays, a scheduled evening out with friends, and small holiday periods) a very long time in advance, it is very unpleasant for him. Hence $k$ should even not be bigger than 8 . Finally, it is more practical to have $k$ even, so you can use constraints which hold for half of the number of weeks.

Later, it became apparent that $k=6$ week scheduling was still the most preferred, since some constraints (e.g. the number of night shifts and work weekends) have been designed for this roster
period length and the department was used to these schedule sizes.
Although automatic rostering software called 'Monaco P.I.P.' is available in the UMC, the roster designer considers the resulting rosters as inappropriate. Hence the schedule is fully created by hand, using the Monaco software as a drawing board, with as a result that the roster designer needs 4 to 5 days per six weeks block to draw the scheme.

The following sections will provide more details about the problem, divided into different information categories. The information is based on several appointments with the unit team leaders and completed questionnaires.

The final sections of this chapter will describe the main goal of the thesis project and the thesis structure.

### 1.1 Employee qualifications

Caring for patients includes different kinds of tasks. Each task involves a certain responsibility level and competence demand. Before an employee may perform a certain task himself, he must have succeeded a training period to become qualified for that task. In this hospital, the employee does not train a single task, but he trains a predefined set of tasks in parallel under supervision during a certain period, after which he will be awarded with a qualification degree. From that moment, the employee is allowed to perform the tasks corresponding to the qualification degree without supervision.

The nurses in this hospital all have their own predefined sets of allowed tasks. We will call such a set a qualification. A nurse can have multiple qualifications, but naturally it is not possible to train for a certain qualification without having a certain other qualification as a base.

For each shift during a day, there is a permanent minimal amount of employees with a certain qualification that should work in that shift. Since the hierarchy of qualifications is roughly linear (an employee with a higher qualification has also a lower qualification, with in rare cases an exception) it is possible to fill up 'low qualified positions' with higher qualified employees. This is not a problem.

The different specialties will be discussed in this section. The abbreviations between brackets refer to the internal (mostly Dutch) name of the qualification, which are provided for clearness and simplicity.

### 1.1.1 Student Nurse (STD)

A Student Nurse is a person who follows an internship in the final stage of his education. The student is not considered competent to make decisions and the major part of his tasks will be done under supervision of other staff members. Furthermore, students are not permitted to work in the night shifts. After graduation, the student will become a basic nurse.

In fact, a student is a nurse in training who has no official qualifications yet. He is treated as a supportive employee and will mostly be added to a shift as surplus. It is preferred to spread the students evenly over the possible shifts.

Last year, the number of Student Nurses fluctuated from time to time. Depending on the roster period, there are approximately $6-11$ Student Nurses available.
[ The abbreviation stands for 'Student']

### 1.1.2 Basic Nurse (VPK)

A Basic Nurse is an employee who is competent to perform most of the basic tasks. Naturally, the nurse has only this single qualification for some time: he will be trained to become a Medium

Care Nurse.
There are approximately 13 Basic Nurses available.
[ The abbreviation stands for 'Verpleegkundige']

### 1.1.3 Medium Care Nurse In Training (MCIO)

This is just an inbetween status and concerns only a few employees. In fact, the nurse is still treated as a Basic Nurse (hence also counted there), who will be trained by a supervisor-colleague for two weeks. After this period, the nurse will occasionally work together with his supervisor until the training has been completed.
[ The abbreviation stands for 'Medium Care In Opleiding']

### 1.1.4 Medium Care Nurse (MC)

A Medium Care Nurse is an employee who is competent to perform some more specialized tasks. Furthermore, he is allowed to perform all tasks a Basic Nurse may perform. This qualification holds for the biggest part of the personnel.

There are approximately 25 Medium Care Nurses available.
[ The abbreviation stands for 'Medium Care' ]

### 1.1.5 Senior Nurse (SVK)

A Senior Nurse is a Basic Nurse with a kind of manager capabilities. He is allowed to perform all tasks a Basic Nurse may perform and has in each roster a few so-called "quality days" for training and improvement. Normally, the Senior Nurse has also the MC-qualification, but sometimes exceptions might exist.

Last year, the number of Senior Nurses fluctuated from time to time. Depending on the roster period, there are approximately 5-7 Senior Nurses available.
[ The abbreviation stands for 'Senior Verpleegkundige']

### 1.2 Appointment contracts

There are two types of employee appointment contracts: permanent contracts and flexible contracts.

Most available employees have a permanent contract. Their appointments mainly range from 24 up to 36 hours per week, but most of them work 32 hours, which is considered as 'full time'. In an exceptional case, an employee with a permanent contract has an appointment for less than 24 hours per week.
The small group of flexible contract employees mainly has a small average appointment of approximately two shifts per week.

### 1.2.1 Permanent contract

In a permanent contract, the employee has a predetermined amount (h) of hours to work per week. The actual amount of work hours he will obtain per week may vary a bit between different weeks, as long as it does not deviate too much from the average number of hours. At the end of a full schedule of $k$ weeks, the employee must have worked (almost) all the $h \times k$ hours his
contract requires him to work during that period. During a full year, the employee will have the opportunity to consume a determined amount of holidays, in consultation with his supervisors. A holiday replaces a full workday.

Most roster preferences of this employee group will be taken into account whenever possible, but are in general not considered as really hard constrains (although they were stated to be quite hard during a later phase of the project). An exception occurs for approximately $6-8$ preferences per schedule (based on scheduling for six weeks at once) which can be indicated by the employee himself. These preferences must be scheduled according to the demands of the employee, but they must in general be in accordance with employee health policies and some other regulations (e.g. not too much late shifts in a week).

### 1.2.2 Flexible contract

A flexible contract, which is said to be based on the Dutch 'zero-hour contract' system ${ }^{1}$, in principle does not provide any guarantee of employment.

Normally, these employees have an average work commitment and there is a tendency to roster the nurses for this average number of hours per week, whenever possible. However, the employee may be assigned to less shifts in a particular week without any consequence and without the obligation to assign them to more shifts in another week. This is typically done when there are already enough nurses in a particular shift the employee prefers to work.

Hence the employee is treated as a kind of schedule filler and there is a tendency to create his roster only after the permanent employees have been scheduled.
The employees in this group may also indicate the days and shifts of the week they prefer (not) to work. However, their constraints are considered as (rather) hard, since, according to the unit direction, the employee has no obligation to perform the assigned work if he does not want to.

### 1.3 Forwards rotating rosters

Nowadays, more and more attention is given to healthy rostering. The underlying assumption is that a healthy employee will have more energy to perform his work, with better results as a consequence, and he will be less absent due to illness. This is certainly an issue in a $24 / 7$ environment, where the day is generally divided into three shifts: early (also called day), late and night. One possible way to help achieve a healthier work environment then, is the use of so-called forwards rotating rosters.

Definition 1.1 (Shift block). A shift block is a set of consecutive days in an employee's schedule for which the assigned shift (day, late, night, free) is the same. Its length is defined as the number of days that is covered by the block.
In the term shift block, the word 'shift' can be substituted by the name of the shift (e.g. 'day block').

Definition 1.2 (Forwards rotating roster). In a forwards rotating roster an employee will first work a couple of early shifts, thereafter a couple of late shifts, then a couple of night shifts and finally have some days off to rest before the early shifts start again. It is also possible to have some days off on other positions between two shift blocks.

An example of a forwards rotating roster is shown in figure 1.1. The consecutive cells, with the assigned shift inside, represent consecutive days in the employee roster. The length of the shift blocks can be varied, according to the type of working environment. For instance: when the night

[^0]activities demand less available employees than needed during the daytime, the length of a night block will typically be shorter than the length of a day block.

| Day block <br> Length $=3$ |  |  | Late block Length $=2$ |  | Night block <br> Length $=3$ |  |  | Free block <br> Length $=2$ |  | Day block <br> Length $=2$ |  |
| :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: |
| D | D | D | L | L | N | N | N | F | F | D | D |

Figure 1.1: Example of a forwards rotating shift pattern.
Experiments in the past, for instance a huge experiment at Corus by Klein Hesseling et al. $(2010)[15]$, have shown that (quick ${ }^{2}$ ) forwards rotating employee rostering has a positive influence on the working staff. Especially older employees will have more benefits from their new roster.

Unfortunately, the definition of a 'forwards rotating roster' is very strict and in practice not always usable or desired. As an example, it is not possible to have a day off between two early shifts and it is not possible to skip the whole night block. In order to obviate this, a second definition will be presented.

Definition 1.3 (Relaxed forwards rotating roster). In a relaxed forwards rotating roster no employee will work in a shift that starts earlier during the day than the shift he worked the day before. In other words: an employee should have $\geq 15$ hours off between two consecutive work shifts.

Definition 1.4 (Block pattern). A block pattern is a set of consecutive shift blocks in an employee schedule. When a pattern needs to be expressed, it will naturally be done in the fashion $\mathrm{X} \rightarrow \mathrm{Y}$ $\rightarrow \mathrm{Z}$, which means that shift block X is directly followed by shift block Y , subsequently followed by shift block Z.

An example of a relaxed forwards rotating roster is shown in figure 1.2. Here, the block pattern Day $\rightarrow$ Late $\rightarrow$ Free $\rightarrow$ Day $\rightarrow$ Night is represented.


Figure 1.2: Example of a relaxed forwards rotating shift pattern.
The new definition of forwards rotation indeed solves the previously sketched problem. Unfortunately, it may be too simple, since it also allows block patterns like Night $\rightarrow$ Free $\rightarrow$ Late $\rightarrow$ Free $\rightarrow$ Day. This is in fact a backwards rotating roster and all the benefits are lost. Only shift patterns like Late $\rightarrow$ Day are ruled out by our new definition.

Since there are some positive effects on forwards rotating rosters, the UMC likes to use a type of forwards rotation in their employee rosters. The exact form is not strictly defined. Therefore, the following rostering rules will be used in this thesis (except an incidental deviation):

- The employee roster follows the relaxed forwards rotation pattern.
- The pattern tries to stay as close as possible to the definition of a strict forwards rotation pattern.

[^1]From now on, when the terms forwards rotation or forwards rotating rostering are used, those will refer to these two rules.

### 1.4 Default hard restrictions for schedules

This section provides a list of default hard restrictions to schedules, in addition to the regulations mentioned in the previous sections. This means that all ${ }^{3}$ (employee) schedules should satisfy these rules. For clearness, the information will be spread over a few subsections with coherent details.

### 1.4.1 Night shift restrictions in employee rosters

Night shifts are considered as horrible by many employees, although some people like them. Student Nurses may never perform a night shift. Most other employees must perform them occassionaly, depending on the number of available employees in the roster period.

When the ward size is sufficiently large, the employees will not necessarily perform a night shift block in each roster period. The scheduling employee may then indicate some employees who obtain an exemption.

- Night shifts must be performed within a night shift block of given restrictions. The length of the block depends on the day of the week. Night shifts can only occur in the following manner:
- Monday - Tuesday ${ }^{4}$
- Monday - Tuesday - Wednesday - Thursday
- Wednesday - Thursday
- Friday - Saturday - Sunday
- After a block of night shifts, the employee will have a minimum of 48 hours off.
- A rare exception excluded, at most 4 night shifts occur per employee during a full schedule of six weeks.


### 1.4.2 Weekend restrictions in employee rosters

In most work environments, personnel has the full weekend off, but that is not possible in a $24 / 7$ situation. Weekend days are therefore treated as a kind of special days. They introduce the following restrictions:

- A staff member will be allocated with shifts both on Saturday and Sunday or he will be off for the whole weekend.
- During a work weekend, the employee will have the same shift on both days.

For the permanent staff, there are a few more restrictions on the amount of work weekends.

- If the employee has an appointment of $>24$ hours per week, he should work 3 full weekends per six weeks.
- If the employee has an appointment of $\leq 24$ hours per week, he should work 2 full weekends per six weeks.
- When an employee is having a holiday, it is possible to allocate less work weekends in his schedule.

[^2]Example: An employee with a holiday of 3 weeks is not obliged to work in the remaining 3 weekends.

The flexible staff does not have well-defined restrictions on the amount of work weekends. Typically, they will work one or sometimes two work weekends in a roster. The work weekends are spread in consultation with these employees.

### 1.4.3 Other restrictions in employee rosters

This subsection contains some shift restrictions that do not fit in the other subsections.

- When the nurse is a flexible employee, the preferences for (not) allocating a particular shift on a particular day are compulsory.

Example: When the employee does not want to work during weekends, he will never be assigned to work during a weekend.

- Maximal 3 late shifts per employee per week.

Remark: Late shifts are considered as nice by lots of employees, but the required staffing for this type of shift is rather low compared to day shifts. Furthermore, the unit management also wants the people who like day shifts to occasionally perform a late shift.

In practice, a lot of employees who like late shifts will exchange them with employees who do not like them. This is implicitly tolerated.

- Every employee has at least one day off per week. ${ }^{5}$
- At the end of the $k$-week roster, the employee should have worked the number of days demanded by his appointment contract. He must have worked the exact number or a deviation of one day. Each vacation day he has consumed in the roster counts as a workday.
- Each Senior Nurse has at least 2 so-called quality days per roster. This type of duty must be planned during the day shift on weekdays. A maximum (also preferred number) of 2 Senior Nurses can be rostered for this task at a day and these employees will not count for the occupancy demands.
- Medium Care Nurses in Training will have the same schedule as their supervisor during the first two weeks of their training.


### 1.4.4 Restrictions in the total roster

The following restrictions hold for the total roster, that is: the combination of all employee rosters. The restrictions are considered hard, but in 'emergency' situations, they may be relaxed a bit.

- For each possible shift in a week, the minimal occupancy and qualification demands must be met.
(Relaxations: 1 Senior Nurse instead of 2 ; one employee short, but MC demand should be satisfied.)
- Surplus in the employee occupancy is only allowed during the early shifts (preferably during weekdays).
(Relaxation: allow surplus during a late shift, but do not assign extra night shifts.)

[^3]
### 1.5 Default soft restrictions for schedules

The following sets of restrictions define preferences which should ideally be satisfied, but are not forced to be so. There is a distinction between more important restrictions and less important restrictions.

### 1.5.1 More important restrictions

The following restrictions are not obliged to be met, but are certainly desired.

- Take the soft roster preferences of the individual employees into account.

Examples: Preference for day off, preference for a shift on a certain day, preference for the 'amount' of late shifts during a $k$-weeks roster, etc.

- An employee with a permanent contract will work his assigned number of weekly hours in each week ${ }^{6}$. Hence (especially) extra workdays in a week should be prevented.
- The smaller the appointment, the more annoying a single day off will be considered. For small appointments ( $<32$ hours / week) single days off are basically not allowed. For the bigger appointments ( $\geq 32$ hours / week) it is preferred not to roster them, but if they give benefits (like rostering a permanent preference of the employee), it is allowed to do so.
- Work weekends and weekends off should be spread well over the roster of the employee.
- Employee surplus should be leveled over the early shifts, preferably during weekdays.
- When the appointment contract involves $\leq 24$ hours per week, it is preferred to schedule 72 hours off after a night shift.


### 1.5.2 Less important restrictions

These restrictions are not that important, but it would be nice if they can be realized. Most of them will probably already be realized without special effort.

- A forwards rotating shift pattern of Day $\rightarrow$ Free $\rightarrow$ Late $\rightarrow$ Free $\rightarrow$ Night $\rightarrow$ Free is the most preferred one (or without the first or second free block). There are no general soft restrictions for the length of the shift blocks. They depend on the employee appointment sizes and their individual preferences.
- Some employees prefer to have 2 blocks of 2 night shifts, others prefer to have 4 night shifts in a row.
- The late block may have a length of at most 3 shifts, the minimal length again depends on the employee's preferences.
- An employee who prefers late shifts over day shifts will have shorter day blocks. Other employees will have more.
- The night shift blocks of an employee should be spread well over his schedule.
- Student nurses should be evenly divided on the schedule.
- When an employee has a late shift during a weekend, it is preferred to have one on Friday too (which is only seldom satisfied in the current rosters).
- It is not a problem if an employee mainly wants to perform early shifts. He could retrieve them, as long as he will have a late and night shift every now and then.

[^4]- After the first two weeks of their training period, Medium Care Nurses in Training will regularly work with their supervisor.


### 1.6 Minimal occupancy demands

The amount of staff that should be assigned to each shift should satisfy the pre-determined minimal occupancy demands. The demands are the same for both weekdays and weekends. As described in the problem description, a (small) surplus during the day shifts is alright, but it is preferred to occur during weekdays.

The demands of the UMC are defined as follows:

- Day shift:
- Minimum: 11 employees.
- Preferred maximum: 14 employees during weekdays, 12 during weekends.
- At least 5 Medium Care (MC) nurses.
- At least 4 additional nurses with Medium Care (MC) or Basic (VPK) qualification.
- At least 2 assigned nurses should have the Senior Nurse (SVK) qualification. ${ }^{7}$.
- Typically, 1 or 2 Student nurse(s) will also work in this shift (not a hard demand).
- Employee surplus is preferred, as long as it is spread well over the day shifts.
- Late shift:
- Total: Exactly 6 employees.
- At least 3 Medium Care (MC) nurses.
- At least 2 additional nurses with Medium Care (MC) or Basic (VPK) qualification.
- 1 other employee, which may be a Student nurse (STD).
- Employee surplus is not preferred.
- Night shift:
- Total: Exactly 3 employees.
- At least 2 Medium Care (MC) nurses.
- 1 additional Basic nurse (VPK) or Medium Care (MC) nurse.
- Allocation of Student nurses is forbidden.
- Employee surplus is forbidden.

Note that the maximum during the day shift is expressed as 'preferred maximum'. At least 11 employees are needed, but it is no problem when some additional employees are assigned to the shift. It is not desired to assign more employees to the day shift than the preferred maximum.

However, sometimes it happens that every day shift already has the preferred maximum number of employees. In case another employee must then be assigned, it is still needed to allocate another employee to a 'full' day shift, since surplus must especially be assigned to the day shifts.

[^5]
### 1.7 Data analysis

We have obtained different data sheets from the UMC during the project, but the most important ones can be found in the appendices $\mathrm{B}, \mathrm{C}$ and D .

- Appendix C shows a list of employee preferences (or sometimes demands) currently known. The information is designed for a period in February - March 2013, but provides a good indication of the kinds of important employee preferences and demands.
- Appendix D shows an example schedule for a period with full formation during 2012. A small part of the roster (the first and last employees) is outside the scope of the project. A lot of different notation for each type of employee is used, which is declared in the legend table in appendix B.

However, in this thesis will only use the terms 'Day', 'Late' and 'Night' to denote a workshift, quality day or training day of an employee. When the employee does not perform any kind of work on a certain day (due to whatever kind of reason), we will denote it by allocation of a 'Free' shift.

This section will provide a brief analysis of this data (especially the employee preferences) and some irregularities that have shown up.

### 1.7.1 Types of employee preferences

Although different kinds of employee preferences can be distinguished, most of them are pretty much of the same type. The preferences were first presented as "nice if most of them can be satisfied", but it turned out the preferences were actually rather hard demands in general. For example: providing a preferred evening off every now and then is not enough, (allmost) all evenings should be off. The UMC wants to be a good employer and therefore satisfy as many requests as possible.
As already mentioned, the preferences of flexible employees should definitely be satisfied in the final schedule.

### 1.7.1.1 Request for a particular shift

The most occurring kind of employee preference concerns the allocation of a shift on a particular day of the week. These kinds of preferences mostly hold for every week of the roster period. Two types can be distinguished:

1. The employee would like to have a particular shift (or day off) assigned on a day (e.g. a day shift during the weekends). Hence there is only 1 shift preferred to allocate.
2. The employee would like to have a particular shift (or day off) not assigned on a day. Hence the employee does not really care which shift is actually assigned on the day, as long as it is not the unpreferred one.

Many employees do not want to have a late shift on a particular day, mostly because of sport reasons. Most other types of these shift requirements concern the possible shifts to allocate in a workweekend, and whether the night shift block is preferred during weekdays or during the weekend. Sometimes, it is requested to have a particular day of the week off.

A special kind of shift requirement is a training day, which especially occurs in the student nurse's rosters. Most of the time it is a fixed day of the week, on which the student has to go to college. Of course the employee is not available to work in the UMC on this day. Hence, this day must be allocated as a day shift without counting the employee for the occupancy demands on the specified day.

The shift requirements can also occur incidentally: a preference for (not) allocating a particular shift on a particular date. With a possible exception, this is in most cases a preference for an
incidental day off, a holiday (large group of days off) or a particular workshift is requested for some kind of training or supervision. These requests are in general hard constraints.

### 1.7.1.2 Request for the length of a block of workshifts

Many employees have a request for 'max $x$ ' occurrences of a shift type $t$ (e.g. max 3 night shifts). This means the employee would like to have at most $x t$-shifts per week or the employee wants to have the $t$-shift for at most $x$ consecutive days.

This type of preference mostly occurs in relation to night shifts. The employee prefers not to have a block of 4 night shifts. Sometimes, 3 consecutive night shifts is already considered as too much. On the other hand, some employees just prefer to have long blocks of night shifts (3 or 4 consecutive days instead of 2 ).
It is important to note that the max keyword (almost) always refers to the length of the shift block, although it can be read as a maximal shift number for the whole roster. That kind of preference will be discussed in section 1.7.1.3.

### 1.7.1.3 Preference for the frequency of workshifts in a particular roster

Many employees have a shift type they (do not) like. They have a preference to allocate this shift as little or as often as possible. This means the shift should (not) frequently occur in the employee's schedule.

Again, this type of preference typically occurs in relation to night shifts (e.g. the preference ' $3-4$ ND', which means 3 or 4 night shifts during a schedule), but the phenomenons 'as little late shifts as possible' or 'preferably late shifts' also occur.

### 1.7.1.4 Forwards rotating pattern

Sometimes a forward rotating pattern is requested, but this is also the default request of the management.

### 1.7.1.5 Predefined schedules

Some employees do not only have some preferences, but they have a full schedule they prefer for that period. This will be discussed in section 1.7.2.2.

### 1.7.2 Special rosters for employees

Some employees have a (kind of) special roster. This means that some hard or soft constraints will not hold for them. Hence the way in which they get rostered will differ from other people. This subsection will provide a short analysis of these employees.

### 1.7.2.1 Permanent appointed employees with hard preferences

Some permanent appointed employees have made an arrangement, such that some of their preferences are considered as hard constraints. Most of the time, having little children whose days of childcare are predetermined is at the base of such arrangements.
The hard preferences are typically of the following type:

- On a particular day of the week, a specific shift must be assigned.
- On a particular day of the week, a specific shift will never be assigned.

Example: Always Tuesday evening off.

- A particular day of the week will always be off.
- The period of the week for night shifts is predetermined.

Example: When a night shift is scheduled, it must be in the weekend.
In general, these combinations of preferences result in a schedule which is largely predefined. These employees have approximately two or three fixed workdays in a week, and should work a fixed number of weekends. Furthermore, they have some predefined extra workdays during some weeks, to achieve exactly their required number of workdays during a six weeks period.

To be more precise: the schedule is typically divided into $k(=6)$ week-blocks, each of them having its own workload. Each block is an almost predetermined configuration of shifts for a week. Only the order of the weeks is not fixed.

We will call this pattern a predefined weeks pattern, built up from predefined week objects:
Definition 1.5 (Predefined Week). A predefined week is a description of a roster for a single week out of $k(=6)$ weeks together with how many times this week description must occur in the full period schedule. For example: "One week of the roster consists of a day shift on Monday, a day or late shift on Wednesday and two day shifts during the weekend. In the final roster, there are two weeks for this employee which match with this description".
The next example will try to clarify the concept.
Example 1.1 (A typical situation to use predefined weeks). An employee with an appointment size of 3 workdays per week has the following requirements:

- Monday: always a day or late shift.
- Thursday: always a day or late shift.
- Two mandatory work weekends: always day or late shifts.

This employee may then have the following predefined weeks:

- $4 x$ A normal week of only Monday and Thursday.
- 2x A week with a weekend: Monday, Thursday, Saturday and Sunday

Unfortunately, the employee should work $3 \times 6=18$ workdays during 6 weeks and now only has obtained $4 \times 2+2 \times 4=16$ workdays. Hence, there is an agreement that this employee also two times works a day shift on Wednesday, so the set of predefined weeks becomes:

- 2x Monday (day or late) and Thursday (day or late).
- 2x Monday (day or late), Wednesday (day) and Thursday (day or late).
- $2 x$ Monday (day or late), Thursday (day or late), Saturday (day or late) and Sunday (day or late).


### 1.7.2.2 Self-scheduling

Some employees have acquired the 'right' of self-scheduling. Some time before the new schedule is created, an empty one is put in the staff room. Besides indicating the set of hard preferences for the next schedule, some employees also fill in their own roster. Thereafter, the scheduling employee tries to do his best to assign these people according to their preferences. Hereby, if it is necessary for the occupancy rate, he tries to deviate from the self-schedules on at most $\pm 3$ days.

In these predefined rosters, strange shift patterns which are considered as undesired by the management may sometimes occur. For instance: an employee could assign himself the pattern Late $\rightarrow$ Day, a pattern for which the scheduler states he would never assign it himself. Nevertheless, the
scheduler declares that he will assign these patterns in case the employee explicitly wants them. Furthermore, more than the desired 4 night shifts are sometimes requested and satisfied.

### 1.7.2.3 Very special rosters

In a rare case, an employee has a very special roster and is scheduled separately. At this moment, there is one employee who is assigned to all night shifts during one full week (Monday - Sunday) and then has one full week of days off. This person is very easy to roster, since the only two possible rosters can be inserted directly into the computer by hand.

### 1.8 Main goal and assumptions

The main goal of this thesis project could be formulated as follows:
Definition 1.6 (Main goal). Create a technique that is able to create an employee work schedule for the nurses of the department of cardiothoracic surgery of the UMC. Both the provided department and employee preferences and requirements should be satisfied as much as possible.
We assume that the available amount of staff is large enough to create a full roster which satisfies the hard constraints and occupancy demands.

### 1.9 Thesis structure

The thesis is structured as follows:

- Chapter 2 discusses a (small) literature study. It will be researched what kind of solutions (within the field of operations research) have been applied in the past and how applicable these solutions are for the UMC situation.
- Chapter 3,4 , and 5 together discuss the main solution.
- Chapter 3 treats the main approach (combining pregenerated schedules) of the solution.
- Chapter 4 investigates more low-level details on schedule generation.
- Chapter 5 discusses the results of initial tests and introduces some extensions to the main solution. Moreover, a realistic experiment setting is discussed, together with the corresponding results.
- In chapter 6 , we explore methods of automatic resolving (occupancy) problems after the schedule for a roster period has been created. Also this time, an experiment setting is discussed.
- The thesis concludes with a conclusion and discussion in chapter 7. We will discuss the positive and negative features of the presented system and provide pointers for future research.

All input data and the actual results of experiments can be found in the several appendices of this report. An extensive overview can be found in the table of contents.

## Chapter 2

## Algorithmic analysis

In the literature, a lot of research about nurse rostering problems can be found. An early survey by Warner (1976)[27] differentiates three types of solution approaches:
Manual Scheduling: The schedules are fully created by hand. This system is also called Traditional Scheduling. The major benefit is its high degree of flexibility.

For a long time, this approach used to be the default one and even nowadays you will find hospital schedulers who find a computerized solution not sufficient. Hence creation is regularly still performed by hand, as is the case in the UMC problem (see chapter 1).
Nevertheless, it is a very time-consuming task to perform[4] and only a small fraction of possible schedules is considered, which likely makes the result very suboptimal. In a time of cutbacks in health care and a lot of political complaints about large overheads in our health care system, this system will not be tenable in the long-term.

Cyclical scheduling: An ordered set of schedules $(1 \ldots k)$ for a fixed time period has been created in advance. Each of the $k$ employees will start with a different schedule number from the set and perform all its assigned duties during the time period. Thereafter, the employee will retrieve the succeeding schedule for the period thereafter. This process continues until the employee retrieves his start schedule again, whereupon the cycle restarts.
The major benefits of this system are its simplicity and that it generally provides good schedules (from a manager's point of view). All assignable schedules can be made healthy (so ad hoc solutions in incidental cases are not needed), the resulting total schedule definitely meets all the occupancy demands, and since all employees will obtain the same schedules, nobody will be jealous. Furthermore, the employees know their schedule a long time in advance, since the same blocks are used repeatedly. Hence, it is easier for them to make plans on the long term.

However, it is difficult or even impossible to take personal preferences into account, which makes the schedules not very satisfactory for the concerned employees. Furthermore, cyclical schedules will only work well when the occupancy demand is pretty stable and the assumption can be made that there is enough staff available to perform the unpopular shifts (Hung, 1992)[18].

Computer Aided Scheduling: The traditional scheduling process is (partly) done by a computer system. The benefits of this system are high: if it works well, the personal staff schedules are both healthy and personalized to individual preferences. Furthermore, the search process takes more solutions into account. It is still impracticable to find the best possible schedule (if at all possible to define it), but a good suboptimal solution which meets the hard constraints is already satisfactory.

After the computer mathematics, the schedule can manually be tweaked whenever desired.

Since the first solution is currently used in the UMC and the second one is undesired for numerous reasons, this project will only focus on Computer Aided Scheduling.

A major overview paper by Burke et al. (2004)[4] considers the process of hospital employee scheduling as "particularly challenging, because of different staffing needs on different days and shifts". He en Qu (2009)[13] state that nurse rostering problems entail a large number of specific constraints, which makes the problems often over-constrained and hard to solve efficiently. Solving them is said to be $N P$-hard[13], which means that it is unlikely that a polynomial time exact algorithm will be found. It is even stated that "nurse rostering is more complex than the traveling salesman problem" $[4,25]$.

From a practical point of view, the complexity would not have been a big issue when the problem sizes were small. However since the search space grows exponentially when the problem size increases, exact algorithms, including constraint programming, are considered to be computationally too expensive[13]. Another common difficulty with rostering problems is that "the quality of a solution is not necessarily a sum or a combination of the qualities of the partial solutions" $[5,9]$. Finally, the objective function (how should we optimize the problem) is often very unclear.
Different techniques have been applied during the last 4 decades. Burke et al. (2004)[4] provides a nice overview of them.

Two kinds of techniques from the field of operational research will be discussed in more detail: local search and linear programming. Furthermore, we will analyze one particular paper of the latter technique in more detail.

### 2.1 Local search

A lot of the already researched solution possibilities include techniques which are based on heuristics and local search methods. The main idea of local search is to take a possible (likely bad) solution to the problem as a start, whereafter it is slowly modified according to predetermined rules, such that new and hopefully better solutions will be created.

In its most default form, the local search process is a hill-climbing algorithm. Using a predetermined objective function, every possible solution obtains a value to indicate its quality. After every modification, the solution will be evaluated using the objective function. Only when its value is higher, the new solution will be kept and used as a base for the next modification.

The improvement continues until no modification yields a better objective value. Hence a local maximum is obtained: a solution which can not be made better with the aid of the predetermined rules (neighbourhood operations). However, this solution does not need to be the best possible solution to the problem (the global maximum).

Figure 2.1 gives a simple two-dimensional state landscape to illustrate the process. The current solution is modified in the direction of the arrow, until a local maximum is reached.

Important variations on hill climbing are tabu search and simulated annealing. Russel and Norvig (2003)[24] provide a good overview of various local search techniques.

### 2.1.1 Examples

Dowsland (1998)[8] provided an iterative three-stage tabu search algorithm for a rather simple variant of the problem. For each employee, a possible roster with a certain cost (reflecting how well it satisfies the employee's preferences) is constructed from a set of natural shift patterns. The basic neighbourhood of a solution is defined as all solutions with a change of a pattern in one of the rosters (instead of a change of a single shift). The three stages consist of several steps.

In the first stage, an initial basic solution which satisfies the coverage constraints is created.


Figure 2.1: Illustration of the landscape in a hill-climbing process. Whether the global maximum will be found, depends on the initial solution (state). The figure is derived from Russel \& Norvig (2003)[24].

Starting from a completely randomly generated solution ${ }^{1}$, it is first tried to satisfy the coverage constraints by applying some basic neighbourhood operators. Only moves that lead to a solution with a lower cost are considered. Thereafter, in the second step of this stage, the basic moves are alternated with more complex chain operators. A chain operator is defined as a sequence of particular moves. The individual moves will not create a better solution (and are therefore never applied in the first step of the stage), but all of them together indeed will improve the solution. Possible chains are calculated using a graph algorithm.

The second stage deals with optimization: try to exchange parts of the schedules between nurses, such that the individual schedules will improve (and the costs will decrease), while retaining the coverage of all shifts. The last stage is used for a strategic disturbance of the results: a move that increases the solution costs is selected, such that the first stage can restart again to hopefully find better solutions.

As already said, the underlying problem is rather simple. The constraints are mainly coverage ones and there is only paid little attention to individual employee roster quality. It is noteworthy that the constraints of the overall occupancy problem are formulated as an Integer Linear Program (see section 2.2), although no ILP techniques are used for solving it.

Burke et al. (1998)[3] applied a hybrid tabu search algorithm to nurse rostering problems based on situations in Belgian hospitals. Their technique was used in a former commercial schedule tool called 'Plane'.

Since their only formulated hard requirement is the fit of the occupancy demand and the big problem is split up into several small subproblems (groups of people with the same qualification, each treated independently of other groups for calculating rosters), it is not difficult to calculate a feasible initial solution. Thereafter, a tabu search algorithm is used to obtain a solution that deals better (as expressed by an objective function) with the (undefined) list of soft requirements. The algorithm only considers moves of a duty from one person to another on the same day and selects the best possible move, until no more move is possible.

Then a diversification step is applied, which is either 'completing weekends' or an exchange of parts of the rosters of the worst scheduled employees. After this step, the tabu search is performed again until some predefined stop rule has been met.
In the end, a finalization step is applied. The most valuable one seems to consider all possible exchange moves for every pair of nurses and repeatedly perform the best one. However, this is very time-consuming.

[^6]The complete approach has been drawn in Figure 2.2.


Figure 2.2: Hybrid tabu search algorithm by Burke et al. (1998)[3].
The solution looks promising, but is unfortunately designed for and has only been tested in relatively small hospitals with approximately 20 employees in a unit. Since the employees have been classified in qualification groups to reduce the complexity of the problem, the subproblems are even smaller. The chosen modifying operators are already claimed to be very expensive on such small problems, thus it is unlikely that the solution will work for more complex problems. Besides this problem, the early split up of employees into groups with the same qualification can easily lead to suboptimal solutions.

A follow-up paper by the same authors[5] also stated that planning each qualification category separately may prevent their algorithm from finding good quality solutions for some categories. Furthermore, they admit that their developed algorithms are not robust enough to handle difficult problems well and finally state: "Unacceptable solutions usually arise when the constraints on the problem are contradictory. It is then very hard to find the very narrow valleys in the solution space, which contain good schedules."

Therefore, the authors have refined their algorithm using various implementations of a genetic algorithm. The algorithm is called 'memetic' (evolution by imitation). Again, the problem size is only about 20 people. Furthermore, the benefits of getting higher quality solutions are at the cost of increased computational time. Since the chosen operators become more and more expensive as the staff size increases, it is likely that this algorithm is also not an option for bigger environments.

In later papers, the authors decided to try different (also more complex) techniques. For instance one in cooperation with the big optimization company Ortec[6].
Bellanti et al. (2004)[2] have developed an approach in which they use both tabu search and simulated annealing in a largely similar problem as the UMC situation. First, an initial solution is created by a heuristic: the best candidate (defined as the nurse who has currently been assigned to the least number of shifts of the particular type) is selected to be assigned to a specific shift, whereupon the initial solution is completed by setting all the remaining shifts. Thereafter, a set of neighbourhood operators is defined and tabu search or simulated annealing is applied to improve the solution.

The results look promising, but also this solution fails in the sense that only little attention is paid to individual employee preferences. Mainly preferred days off can be expressed and special rosters are not considered at all. Furthermore, the system considers every nurse as having the same qualifications, but in the UMC case nurses could have several qualifications. A problem decomposition, such that each qualification is solved independently, is therefore not possible, since qualifications can overlap and a more skilled employee can replace less skilled ones. Hence the solutions are likely to become too suboptimal in the UMC case.

More algorithms have been developed, but many of them are declared to be too simple or have not been tested in practice.

### 2.1.2 Conclusion

Local search methods can be effective when the ward size and the number of constraints are small. Heuristics will create an initial solution, which is then repeatedly improved.

Moving single shifts around usually does not improve the solution state that much and certainly will not lead to the best solutions. Therefore, more complex neighbourhood operators are needed, which may be hard to find or to define, especially in big constrained situations. Another problem, especially when a large amount of employees is involved, is the expensiveness of the operators that seem to work well.

However, local search may possibly be used to try to improve solutions for the nurse scheduling problem which are calculated using other techniques. This could also be the case for the UMC problem, but it does not appear to be the kind of technique to build the whole solution.

### 2.2 Linear (and constraint) programming

Linear programming (LP) is a mathematical method to find the optimal solution of an objective function, given a set of linear constraints. A problem consists of five important ingredients:

1. A vector of $n$ variables: $\mathbf{x}=\left[x_{1}, \ldots, x_{n}\right]$
2. A cost vector $\mathbf{c}=\left[c_{1}, \ldots, c_{n}\right]$
3. An objective function: minimize or maximize the outcome of

$$
\begin{equation*}
\sum_{j=1}^{n} c_{j} x_{j} \tag{2.1}
\end{equation*}
$$

4. A matrix $\mathbf{A}$ of size $m$ by $n$ and a vector $\mathbf{b}=b_{1}, \ldots, b_{m}$ to define a set of $m$ linear constraints on the $n$ variables:

$$
\begin{equation*}
\sum_{j=1}^{n} a_{i j} x_{j} \leq b_{i} \text { for } i=1, \ldots, m \tag{2.2}
\end{equation*}
$$

5. A lower bound and upper bound restriction on the values of the variables:

$$
\begin{equation*}
l_{j} \leq x_{j} \leq u_{j} \text { for } j=1, \ldots, n \tag{2.3}
\end{equation*}
$$

All constraints and variable restrictions together define a feasible region of the problem, which is a convex polyhedron (see Figure 2.3 for an example).

A large range of algorithms has been developed to find the point in the polyhedron for which the objective value is minimum or maximum (if such a point exists). Possibly the most well-known is Simplex [1], which is quite efficient in practice, although it has poor worst-case behavior. The number of steps can be exponential in the input size.


Figure 2.3: Convex polyhedron representing a simple linear program with 2 variables (number of dimensions) and 6 inequalities (black lines). The feasible region is colored red. Adapted from Wikipedia[29].

A polynomial-time algorithm is Karmarkar's 'interior point' algorithm[21]. Although its worstcase behavior is better, it does not meet the efficiency of Simplex in practice, since larger computational times were experienced in many numerical experiments (Ferris and Philpott, 1988[10]).

A perfect algorithm to solve LP's has not been found yet. All algorithms perform some kind of poor worst-case behaviour or are less efficient in practical situations. Furthermore, it is unknown whether a strongly polynomial-time algorithm for solving LP's exists[29].
In its basic form, the solution of the linear program may contain variables with a fractional value. When only integral solutions are desired, the problem becomes an Integer Linear Program (ILP), which is much harder to solve. In fact, it has already been shown in Karp's 21 NP-Complete problems [22] that the decision variant of an ILP in which the only possible values for the variables are 0 and 1 (0-1 Integer Programming) is NP-hard. Techniques that have been developed to find a solution for an ILP include the cutting-plane method, branch and bound, branch and cut and branch and price[29].

A Mixed Integer Program (MIP) is an LP in which some of the variables are restricted to integral values. In general, these problems are also NP-hard.

The various algorithms are used in a large set of developed computer solvers. Besides an optimization algorithm and solving techniques for ILP's, most of them also contain other techniques to speed up the solving process. A very popular one is the IBM ILOG CPLEX Optimizer ${ }^{2}$, or CPLEX in short.

Most exact ${ }^{3}$ LP solutions for nurse rostering problems do not deal with a lot of real life requirements and therefore they are not worth to highlight. An overview is provided in Burke et al. (2004)[4].

### 2.2.1 Revised Simplex and Column Generation

As described, an ILP essentially consists of a set of variables, each having its own cost. Depending on the practical background of the problem, it may represent a decision problem. A subset $X^{\prime} \subseteq X$ of variables then represents an object, task, pattern ${ }^{4}$, etc., depending on the problem. The ILP determines which of them will be selected or bought (or sold, when a variable $x_{j}$ obtains a negative value) and how many times.
A special situation arises when the values of this subset can only be 1 or 0 (selected or not). The ILP then represents a binary decision problem.

[^7]
### 2.2.1.1 The knapsack problem

A famous illustrative example of a binary decision problem is the Knapsack problem[7, 16].
There is a set of $n$ distinct items $j=1, \ldots, n$. Each of them has a value $c_{j}$ and can be selected to be put in a knapsack which may hold at most $b$ kilograms (the bag size). When an object $j$ is taken, an amount of $w_{j}$ kilograms is added to the knapsack. The purpose is to find a selection of items such that the total value of selected items is as high as possible, but their total weight may not be larger then $b$.

The problem can be written as an ILP:

$$
\begin{equation*}
\text { Maximize: } \sum_{j=1}^{n} c_{j} x_{j}\{\text { Total value }\} \tag{2.4}
\end{equation*}
$$

Subject to:

$$
\begin{align*}
& \left.\sum_{j=1}^{n} w_{j} x_{j} \leq b \text { Enough space }\right\}  \tag{2.5}\\
& x_{j} \in\{0,1\}, \text { for each } j=1, \ldots, n\{\text { No partial selection of objects }\} \tag{2.6}
\end{align*}
$$

Suppose the number of available items in the problem above is very large and it is not possible to take all of them in the bag. We then have to make a selection, but the binary decision constraint (2.6) makes the formulated program an ILP, for which solving is NP-hard (see previous section).

In this section, we will provide some basic techniques which try to improve the ILP optimization process. They can be applied to a wide variety of ILP problems. We will use the knapsack problem as a 'toy problem' to illustrate the concepts being discussed.

The problem can first be relaxed by rewriting equation (2.6) to $0 \leq x_{j} \leq 1$. This is called an $L P$ relaxation. Furthermore, an initially empty set $S$ will be created in which we store a selection of items. While solving the LP, the items in $S$ are the only ones considered for solving the problem. The set $S$ will iteratively be extended with items which are the 'most promising' for finding a better objective value.

The resulting LP problem, also known as the fractional knapsack problem, can be written as:

$$
\begin{equation*}
\text { Maximize: } \sum_{j \in S} c_{j} x_{j} \tag{2.7}
\end{equation*}
$$

Subject to:

$$
\begin{equation*}
\sum_{j \in S} w_{j} x_{j} \leq b \tag{2.8}
\end{equation*}
$$

$$
\begin{equation*}
0 \leq x_{j} \leq 1, \text { for each } j \in S \tag{2.9}
\end{equation*}
$$

The set $S$ is then filled with some of the items using a heuristic or some other method ${ }^{5}$ and the LP is solved for this selection of items. Unfortunately, this solution is not optimal. Hence it can be effective to add another item to $S$ so that a solution which generates more money may be computed.

[^8]
### 2.2.1.2 Shadow Price and Reduced Cost

Before we will discuss how the item that will be added to the problem will be determined, the concepts of shadow price and reduced cost will be introduced.

Definition 2.1 (Shadow Price). The shadow price of a constraint is "the instantaneous change per unit of the constraint in the objective value of the optimal solution"[28]. Or in other words: "the unit price you want to pay for a little more"[16].

In a business application, its equivalent can be defined as "the maximum price that management is willing to pay for an extra unit of a given limited resource" $[28]$.

Example 2.1. Consider the fractional knapsack problem. Suppose $b=4$ and we have the following subset $S$ :

| $j$ | 1 | 2 | 3 | 4 |
| :--- | :--- | :--- | :--- | :--- |
| $c_{j}$ | 8 | 4 | 5 | 2 |
| $w_{j}$ | 1 | 2 | 3 | 4 |

In the optimal solution $\sigma$ : $x_{1}=1, x_{2}=1, x_{3}=\frac{1}{3}$ and $x_{4}=0$. The objective value (total cost) will be $13 \frac{2}{3}$ euros.
Suppose the bag size constraint is increased by a small amount $\epsilon$. For this example, a value $\epsilon=1$ is sufficient, so in the new situation $b=5$. It is then possible to take an additional $\frac{1}{3}$ part of item 3. Hence in the new optimal solution $\sigma^{\prime}: x_{1}=1, x_{2}=1, x_{3}=\frac{2}{3}$ and $x_{4}=0$. The new objective value (total cost) will be $15 \frac{1}{3}$ euros, an increase of $\Delta(\epsilon)=1 \frac{2}{3}$ euros. This entails that the shadow price of the bag size constraint in solution $\sigma$ is:

$$
\begin{equation*}
\frac{\Delta(\epsilon)}{\epsilon}=\frac{1 \frac{2}{3}}{1}=1 \frac{2}{3} \tag{2.10}
\end{equation*}
$$

In fact, to determine the shadow price of a constraint, no recalculations have to be performed. The shadow price is simply a by-product of the LP-solution. It can be found by looking at the parameters of the corresponding dual problem. Details about duality can be found in (for instance) Bazaara et al. (1990)[1].
The second concept is the reduced cost of a variable that is currently not used in the problem set, but is possible to add.

Definition 2.2 (Reduced cost). The reduced cost of a currently unused variable $x_{0}$ is the net gain per $\epsilon$ when a small fraction ( $\epsilon$ ) of the item represented by it is used in the solution. In other words: it is the net gain per $\epsilon$ it would entail when $x_{0}$ is added to the problem set and the optimal solution is recalculated, with $x_{0} \leftarrow \epsilon$.

The value of $\epsilon$ has to be chosen in such a way that only one value $x_{j}$ of an existing item $j$ is changed in the solution. In both example 2.1 above and example 2.2 below, only the value $x_{3}$ is changed.

Example 2.2. Consider the initial situation of example 2.1 (so $b=4$ ) and suppose the set $S$ is extended with a new item $j=5$ with $c_{5}=10$ and $w_{5}=2$. Hence, when a small fraction $\epsilon$ of the new item is used in a recalculated solution, its direct gain to the solution is $\epsilon \cdot c_{5}=\epsilon \cdot 10$ euros.
Unfortunately, because of the bag size constraint, some previously selected items will not be taken any longer. Hence the optimal value will not increase by $\epsilon \cdot 10$ euros.

Let $\epsilon=\frac{1}{2}$ for this example, so the direct gain is 5 euros. In fact, in a new optimal solution $\sigma^{\prime \prime}$ we have: $x_{1}=1, x_{2}=1, x_{3}=0, x_{4}=0$ and $x_{5}=\frac{1}{2}$. The new objective value (total cost) thus will be 17 euros, an increase of $\Delta(\epsilon)=3 \frac{1}{3}$ euros. Hence the reduced cost of adding an amount $\epsilon=\frac{1}{2}$ of item 5 to $S$ is:

$$
\begin{equation*}
\frac{\Delta(\epsilon)}{\epsilon}=\frac{3 \frac{1}{3}}{\frac{1}{2}}=6 \frac{2}{3} \tag{2.11}
\end{equation*}
$$

In the example, the 'investment' per $\epsilon$ of using an amount $\epsilon=\frac{1}{2}$ of item 5 is equal to:

$$
\begin{equation*}
\frac{5-3 \frac{1}{3}}{\epsilon}=\frac{1 \frac{2}{3}}{\frac{1}{2}}=3 \frac{1}{3} \tag{2.12}
\end{equation*}
$$

This is the amount you have to 'pay' for using the item, since some previously selected items cannot be (fully) used any longer, due to the constraints of the problem (the bag size constraint in the knapsack program).

The investment per $\epsilon$ can also be obtained directly from a given solution by looking at the shadow prices of the problem's constraints. It can simply be calculated as the weighted sum of the shadow prices of all the constraints involved when adding the item. In the knapsack program, the only constraint is the bag size constraint and as seen in example 2.1, its shadow price is $\pi=1 \frac{2}{3}$. The weight of item 5 in this constraint is $w_{5}=2$, so the weighted sum of shadow prices for this item is $w_{5} \cdot \pi=2 \cdot 1 \frac{2}{3}=3 \frac{1}{3}$, which is indeed the investment.
Hence it is not needed to recalculate the optimal solution to retrieve the reduced cost of adding an item. The gain of the item is directly provided, the investment is the weighted sum of all shadow prices of the involved constraints and the reduced cost is simply the difference between these two values.

### 2.2.1.3 Performing column generation

In general: Suppose a problem has a set of $m$ constraints with shadow prices (or dual multipliers) $\pi_{1}, \ldots, \pi_{m}$. Similarly, a pattern $j$ which has not been taken into account yet has a variable $x_{j}$, with yield $c_{j}$ and a weight of $w_{i j}$ in constraint $i$. The reduced costs of variable $x_{j}$ is then defined as:

$$
\begin{equation*}
c_{j}^{\prime}=c_{j}-\sum_{i=1}^{m} \pi_{i} w_{i j} \tag{2.13}
\end{equation*}
$$

In the knapsack example, it is only useful to add an item if its reduced costs are positive (since the objective function is to maximize the costs). The best item to add is the one with highest reduced costs. Hence, the unused variable with this property can be determined and added to the problem. The addition of the item produces a column in the constraint matrix of the LP. This procedure is called revised simplex.

For most problems, it is impossible to enumerate all the valid patterns that can be added. Then, the subproblem of finding the pattern with the highest reduced costs can be expressed as an (I)LP itself. This pricing problem (or auxiliary problem) can often be solved efficiently using LP techniques or can be casted to another kind of problem for which an effective algorithm exists; for instance: a shortest path problem in a graph. Since the column to add is not known in advance, but is generated by an external process, this procedure is called column generation.

### 2.2.1.4 The solution of the ILP

After the addition, the LP is solved with the increased item set. This procedure of item additions is repeated until there is no item remaining with positive reduced costs. Then, the optimal solution of the LP is found.

Depending on the problem, the LP solution can be casted directly to the ILP solution by rounding, or the ILP solution needs to be calculated or approximated using the earlier mentioned techniques (such as branch and price). The difference between the optimal value of the LP-relaxation and the optimal value of the ILP is called the integrality gap.

### 2.2.2 Examples

Jaumard et al. (1998)[20] have specified an LP solution using column generation. A distinction is made between generation of individual employee rosters and combining them into a total schedule.

Their (complex) master problem concerns the schedule assignment, fulfilling occupancy demands (with the possibility to work a part of a shift) and scheduler's preferences for assigning particular nurses. An auxiliary problem for each individual employee is used to find a good schedule to add to the master problem. Here, a resource constrained shortest path problem (RCSSP)[19] in a layered directed graph is solved. Each path corresponds to a possible column which could be added to the master problem. The auxiliary problem is solved by both a branching algorithm, which eliminates some vertices, and a two-phase strategy for finding the shortest path.

The graphs are built as follows: For each day of the roster, a layer with vertices is created for each possible shift. Vertices in layers $i$ and $i+1$ are connected whenever the represented shift on day $i$ can be directly followed by the shift represented by the vertex on day $i+1$. It is also possible to add an arc between a vertex in layer $i$ and a vertex in layer $j \in X_{i}=\{x \mid x \geq i+2\}$; then shift $a$ is performed on day $i$, followed by $j-i-1$ days off, whereafter shift $b$ is performed on day $j$. Finally, there are two dummy vertices added as a source and a sink for the shortest path problem.

The length of an arc between two vertices is defined by the current dual multipliers from the master problem and several weighted individual parameters (employee preferences, salary, seniority, desirability of the assignment by the scheduler, etc.). A full path from source to sink is a possible employee schedule. It is valid when it satisfies the resource constraints.

The resources $R_{\alpha}$, each with a lower bound and upper bound, are used to enforce patterns. The following kind of resources are distinguished:

1. Workload: The nurse must work a certain amount of shifts every week.
2. Off Weekends: The nurse has groups of weekends off duty and groups of weekends on duty.
3. Rotation: Diverse complex resource variants to force forwards rotation.
4. Consecutive assignments: It is not allowed to work less or more consecutive days than a predefined number.
5. Shift ratio's: There must be enough variance in the roster: all kinds of possible shifts shall be performed every now and then.
6. Holidays: An employee should have some of the days from a specified day set off (e.g. from the set $\{$ Dec. 25th, Dec. 26th, Dec. 31st, Jan. 1st $\}$ ).
A positive property of the solution is its capability of modeling complex constraints. The resource model can likely be extended with a lot of other real world constraints.

Unfortunately, most of the used resource definitions are already quite complex and the authors state that "special care should however be taken while defining the resource windows and the resetting values, since the complexity of the auxiliary problem algorithm is very sensitive to the width of the windows". In addition, there is no algorithm that solves an RCSPP efficiently. Test results in the paper (with the authors' own algorithm) are only based on some preliminary numerical tests; all of them with a small subset of the before mentioned resource constraints. For these inputs, an LP solution is found within an hour, but solving the ILP already takes almost a day.
Improved algorithms and heuristics with good expectations were said to be developed at the time of publication, but no follow-up paper has been published in the past 12 years. It looks like the research in this field has been aborted. Hence no improvements will be expected.

Hoogeveen and Penninkx (2007)[17] have developed a solution for security personnel working in a $24 / 7$ environment. The employees work in three shifts: early, late and night. Their rosters
must be based on a strictly forwards rotating system (see section 1.3):

$$
\begin{equation*}
\text { Day } \rightarrow \text { Free } \rightarrow \text { Late } \rightarrow \text { Free } \rightarrow \text { Night } \rightarrow \text { Free } \rightarrow \text { Day } \rightarrow \text { Free } \rightarrow \ldots \tag{2.14}
\end{equation*}
$$

All people should work 17 shifts per four-week period, and for each shift the minimal occupancy demands must be met.

The soft constraints state that over-booking is preferred on Wednesday morning and the preferences of the employees must be taken into account: their preferred lengths of each shift block and their favorite fixed day off each week.

The solution is solved using an ILP, which selects rosters from a pregenerated set of schedules for every employee. Each set is generated using a strict forwards rotating pattern. The generated rosters obtain a cost value which reflects the satisfaction of the employee. Rosters with too high costs are discarded in advance.

Essentially, the problem their solution is created for is very simple compared to our nurse scheduling problem, since the number of constraints is limited. For example: there are no weekend and night shift restrictions, all generated schedules follow a rather strict rostering pattern, and preferences for days off and (not) allocating a shift on a particular day are only defined as soft constraints (increase the costs of a generated schedule).

It is likely that a lot of preferences are taken into account every now and then, but it is not possible to guarantee the existence of generated rosters in which an employee obtains a fixed or preferred shift on a certain day of every week (instead of some weeks). ${ }^{6}$

Although the solution has been specified and has been suitable for a real world problem, the lack of hard constraints and the simple schedule pattern makes it unsuitable for most of the real-world nurse-scheduling problems. It is possible to modify the solution method, such that it is able to deal with more constraints, but it remains to be seen how robust the approach will be in practice. As we will see in this thesis, the challenge with pre-generated rosters is that it becomes difficult to find a setting which still finds rosters with enough 'quality' ', while not having an incalculable amount of them. Besides, the solution has never been tested in a working environment, since the management of the hospital decided to restructure the department shortly after the solution was computed.
However, it is interesting to research whether the solution ideas as presented in the aforementioned paper can be extended such that the constraints of the UMC nurse-scheduling problem will be satisfied. Since the solution approach forms a basis for our solution (chapter 3), we will discuss it in depth in a separate section (section 2.3).
Finally, a very recent paper by $\mathbf{H e}$ and $\mathbf{Q u}$ (2012)[14], officially published in December 2012, is worth it to be mentioned. This paper became available online in spring 2012, when experiments with our solution approach were already performed. Although there are some differences, the main solution approach in the paper is quite similar to ours (see chapter 3), where column generation and depth first search techniques for creating good columns are combined. Furthermore, this research[14] is based on a previous publication (He and $\mathrm{Qu}(2008)[12]$ ) in which real-world problems in intensive care units at a Dutch hospital were treated.

Hence, comparing to our UMC problem, there is a lot of overlap in the context of both problems. We will make a few comments about these two papers in our result discussion (section 7.2).

### 2.3 A basic approach by Hoogeveen and Penninkx

This section will discuss a slightly simplified variant of the basic solution of Hoogeveen and Penninkx (2007)[17] for the problem that is described in section 2.2. The full solution and a discussion about the choices that are made can be found in the specified paper.

[^9]The solution for a given time period consists of a main ILP, that uses a set of feasible rosters $S_{j}$ for each employee $j=1, \ldots, n$. At this moment, a set $S_{j}$ is assumed to contain all possible feasible rosters for employee $j$. A roster $s$ for employee $j$ has a cost $c_{j s}$, reflecting how well it meets the employee's preferences (on a scale between 0 and 1 ). The constant $a_{i j s}$ reflects whether shift $i$ is covered in the roster. The variable $x_{j s}$ reflects whether roster $s$ is selected for employee $j$.
Furthermore, each shift shall preferably not be over-booked and certainly not be under-booked. The costs of over-booking and under-booking of shift $i=1, \ldots, m$ are defined as $f_{i}$ and $g_{i}$ respectively. The deviations from the optimal occupation are defined as $y_{i}$ and $z_{i}$ respectively.

Hence the objective is to minimize the total cost, which is the cost reflecting the extent to which the employee preferences are granted or ignored in the selected rosters and the cost of over-booking and under-booking:

$$
\begin{equation*}
\text { Minimize: } \sum_{j=1}^{n} \sum_{s \in S_{j}} c_{j s} x_{j s}+\sum_{i=1}^{m}\left(f_{i} y_{i}+g_{i} z_{i}\right)\{\text { Total cost }\} \tag{2.15}
\end{equation*}
$$

The constraints in their model state that:

- For each shift $i$ the occupation demand $b_{i}$ is met, or over-booking/under-booking is penalized (equation (2.16)).
- Each employee $j$ retrieves exactly 1 roster (equation (2.17)).
- An employee roster is fully selected or not at all (equation (2.18)).
- Over-booking and under-booking sizes can not be negative (equation (2.19)).

This results in these linear equations:

$$
\begin{align*}
& \sum_{j=1}^{n} \sum_{s \in S_{j}} a_{i j s} x_{j s}=b_{i}+y_{i}-z_{i}, \text { for each } i=1, \ldots, m\{\text { Demand }\}  \tag{2.16}\\
& \sum_{s \in S_{j}} x_{j s}=1, \text { for each } j=1, \ldots, n\{\text { Employee gets } 1 \text { roster }\}  \tag{2.17}\\
& x_{j s} \in\{0,1\}, \text { for each } s \in S_{j}, j=1, \ldots, n\{\text { No partial roster selection }\}  \tag{2.18}\\
& y_{i}, z_{i} \geq 0, \text { for each } i=1, \ldots, m\{\text { No negative over-/under-booking }\} \tag{2.19}
\end{align*}
$$

To solve the ILP, it has first been relaxed to an LP by replacing $x_{j s} \in\{0,1\}$ in the constraints 2.18 with $x_{j s} \geq 0$ (the upper bound is not enforced, since it will automatically follow from the constraints 2.17). The LP is then solved using column generation on a restricted set of rosters $Q_{j} \subset S_{j}$, for each $j=1, \ldots, n$ : the best rosters with respect to the optimal solution of the LP-relaxation.

### 2.3.1 Creation of the sets $Q_{j}$

For each employee, a set of possible valid rosters (according to the shift pattern regulations) for a 4 week period is generated by enumeration. Each roster obtains a cost $c_{j s} \in[0,1]$, reflecting how well it meets the employee's preferences. Then, a pre-selection is made by dropping all generated schedules with costs $c_{j s}>0.5$.
The sets $Q_{j}$ of rosters that are used to iteratively solve the main LP-problem are determined by column generation.

Initially, a set of good rosters is added for each employee and the LP is solved using CPLEX (see section 2.2). Then, the still unused rosters with the lowest reduced cost $c_{j s}^{\prime}$ from the set with generated rosters are selected for each employee and added to the problem. This process continues iteratively until no good roster can be added ( $c_{j s}^{\prime} \geq 0$ for all unused rosters for all employees) or a time limit has been reached.

The pricing problem is defined as finding the rosters with the lowest reduced cost:

$$
\begin{equation*}
\text { Minimize: } c_{j s}^{\prime}=c_{j s}-\lambda_{j}-\sum_{i=1}^{m} \pi_{i} a_{i j s} \tag{2.20}
\end{equation*}
$$

The parameters $\pi_{1}, \ldots, \pi_{m}$ and $\lambda_{1}, \ldots, \lambda_{n}$ are the dual multipliers corresponding to the constraints (2.16) and (2.17) respectively; the variables are $a_{i j s}$.

Since the problem is solved for each employee independently, it can be simplified to:

$$
\begin{equation*}
\text { Minimize: } c_{j s}^{\prime}=c_{j s}-\sum_{i=1}^{m} \pi_{i} a_{i j s} \tag{2.21}
\end{equation*}
$$

### 2.3.2 Calculation of follow-up rosters

The approach for 4 weeks rosters can be extended to calculate rosters for longer periods, consisting of an integer number of blocks of 4 weeks. Naturally, the consecutive rosters should follow-up well.

Since the number of possible 4 weeks rosters for an employee is relatively small (due to the very strict shift pattern obligation), it is possible to use 8 weeks rosters in the (I)LP. These rosters are combinations of two 4 weeks period rosters and are calculated during the column generation phase using a layered graph of all possible roster combinations for every employee $j$. The graph has four layers (figure 2.4):

1. A start node $s_{j}$, representing the last 4 weeks of the current roster for employee $j$.
2. A layer $\mathcal{A}_{j}$ representing the first period, with nodes for all possible rosters following the current roster. All rosters have an incoming arc from $s_{j}$.
3. A layer $\mathcal{B}_{j}$ representing the second period, with nodes for all possible rosters following a roster in layer $\mathcal{A}_{j}$. If it is possible to combine a roster in layer $\mathcal{A}_{j}$ with a follow-up roster in layer $\mathcal{B}_{j}$, an arc is initialized from the first to the last.
4. A final node $t_{j}$. All rosters in layer $\mathcal{B}_{j}$ have an outgoing arc to $t_{j}$.


Figure 2.4: Layered graph network to produce consecutive rosters.
Each outgoing arc of a node in layer $\mathcal{A}_{j}$ and $\mathcal{B}_{j}$ retrieves a value equal to the reduced cost of the 4 weeks roster the node represents. A shortest path is then calculated between $s_{j}$ and $t_{j}$ using a default single-source shortest path algorithm (see Cormen et al. (2009)[7] for an overview). An existing path represents a valid follow-up roster. The 'distance' represents the total reduced cost.

Hence this approach selects for each employee a feasible 8-week follow-up roster with lowest reduced costs. This roster can then be added to the master LP-problem as a column and the LP and subsequently the ILP can then be solved.

### 2.3.3 Solving the ILP

After the optimal value of the LP has been calculated, the ILP is solved using the various techniques available in CPLEX. The calculation is done with the use of all the rosters that were added to the LP, together with an additional 2500 most-preferred rosters (based on the dual multipliers in the optimal LP solution).

### 2.3.4 Calculation of rosters for a large roster period

Finally, when a large roster period needs to be calculated, it is possible to extend the 8 -week approach from the previous section for longer periods, consisting of an integer number of blocks of 4 weeks. For the LP-phase, it is no problem to calculate all feasible rosters for a full year (52 weeks). Unfortunately, the ILP-phase encounters major problems.
To solve the ILP for a large roster period, a so-called rolling horizon approach can be applied. This works as follows:

1. The graph node $s_{j}$ is undefined on initialization. Calculate an initial roster for 8 weeks (two 4 -week periods) for each employee by solving the LP and subsequently the ILP. Every start position of an employee roster is applicable.
2. Fix the first 4 weeks of the previously calculated roster for each employee and define $s_{j}$ as the node representing this roster (the last 4 weeks of the calculated roster are not used). Calculate a consecutive roster for 8 weeks (two 4 -week periods) for each employee by solving the LP and subsequently the ILP.
3. Repeat step 2 until the roster for the full period has been calculated.

## Chapter 3

## The main approach

In chapter 2, we discussed the solution approach by Hoogeveen and Penninkx (2007)[17] to solve the security personnel scheduling problem. Their solution considers two subproblems:

1. Find possible nice schedules for each individual employee.
2. Combine the employee schedules into a final total schedule. Select a good roster for each employee, such that the occupancy demands become satisfied.

We will investigate the extent to which this approach can be modified and extended, such that it satisfies the UMC nurse rostering problem constraints. We will discuss our solution from a top-down approach.
This chapter will consider the second subproblem, which also deals with qualification constraints on the occupancy demands. We will call this our main problem. The main problem is solved using linear programming in combination with column generation to introduce the individual schedules.

The subproblem of finding nice individual employee schedules will be discussed in chapter 4.

### 3.1 Legend of constants and variables

Throughout the next sections, a lot of mathematical variables and constants will be used. This section will provide an overview of the most important ones. Sometimes, a section will provide a few additional constants or variables.

### 3.1.1 Primitives

$i$ A particular shift on a particular day $(i \in I)$.
$j$ An employee $(j \in J)$.
$m$ Total number of shifts $(m=|I|)$.
$n$ Total number of employees $(n=|J|)$.
$q$ A qualification $(q \in Q)$.
$s$ A full individual employee schedule for $k$ weeks $(s \in S)$.

### 3.1.2 Occupancy demands

$b_{i}$ The minimal number of employees needed for shift $i$.
$b_{i}^{q}$ The minimal number of employees with qualification $q$ needed for shift $i$.
$o_{i}$ The maximal number of employee surplus in shift $i$ that is still acceptable. ${ }^{1}$

### 3.1.3 Schedule properties

$a_{s j}$ Whether schedule $s$ has been designed for employee $j$.

$$
a_{s j}= \begin{cases}1 & \text { when schedule } s \text { has been designed for employee } j \\ 0 & \text { otherwise }\end{cases}
$$

$d_{s i}$ Whether shift $i$ is covered in schedule $s$.

$$
d_{s i}= \begin{cases}1 & \text { when shift } i \text { is covered in schedule } s \\ 0 & \text { otherwise }\end{cases}
$$

$h_{j}^{q}$ Whether employee $j$ has qualification $q$.

$$
h_{j}^{q}= \begin{cases}1 & \text { when employee } j \text { has qualification } q \\ 0 & \text { otherwise }\end{cases}
$$

### 3.1.4 Cost functions

$c_{s}$ Costs of schedule $s$, reflecting how well it deals with the preferences of the involved employee.
$f_{i}$ Costs per acceptable redundant employee in shift $i .^{1}$
$F_{i}$ Big costs per redundant employee to force an upper limit on the employee surplus in shift $i .{ }^{1}$
$g_{i}$ Costs per employee shortage in shift $i$.
Remark: It should hold that $0 \leq f_{i}<F_{i}$ and $g_{i}>0$.

### 3.1.5 Decision variables

$x_{s}$ Schedule assignment: which schedule is selected for each employee.

$$
x_{s}= \begin{cases}1 & \text { when schedule } s \text { is selected. } \\ 0 & \text { otherwise }\end{cases}
$$

$y_{i}$ The number of employee surplus in shift $i$.
$Y_{i}$ The number of employee surplus above acceptable limits in shift $i .{ }^{1}$
$z_{i}$ The number of employee shortage in shift $i$.

### 3.1.6 Remarks

For every shift $i$, there is a minimal occupancy demand $b_{i}$. It is possible to have some small employee surplus. The acceptable surplus is defined with $o_{i}$.
There are two possibilities to force this acceptable upper bound. The first one is only using a parameter $y_{i}$ with cost parameter $f_{i}$ for the number of employee surplus, which is bounded by a constraint $0 \leq y_{i} \leq o_{i}$. The problem with this constraint is that it will become more difficult to find an initial solution of an (I)LP when revised simplex or column generation is applied, since it is not possible to have a surplus of more than $o_{i}$ in a certain shift.

[^10]To overcome this, an additional variable $Y_{i}$ with a cost parameter $F_{i}$ is introduced. It must hold that $f_{i}<F_{i}$ and $F_{i}$ should have a large value. In this way, a large surplus in a shift in the initial solution is allowed, leading to an easier calculation of the initial solution. First, the surplus 'space' up to $o_{i}$ is consumed with small costs. Once the whole space has been used, large costs will be spent for the remaining employees surplus.
Since the initial solution of the second situation will lead to a very high cost value, it is likely the costs will become lower in later solutions. The final effect is about the same as with the first situation. Furthermore, a nice solution with some small deviations can possibly be modified into another by a human being after the computation has been done.

### 3.2 Master problem with task assignment

A possibility to introduce qualification constraints is by not only indicating which employees will work which shifts, but also specifying the 'task' they will then perform. A 'task' in this context is not a real duty, but a satisfaction of a qualification demand. The assignment is only to ensure that enough employees of each qualification are available during the shift.We will introduce an additional variable:
$w_{i j}^{q}$ Whether employee $j$ is used to satisfy the constraint for qualification $q$ in shift $i$.

$$
w_{i j}^{q}= \begin{cases}1 & \text { when employee } j \text { counts for qualification } q \text { in shift } i \\ 0 & \text { otherwise }\end{cases}
$$

We can now rewrite the ILP discussed in section 2.3 by introducing new constraints:

$$
\begin{equation*}
\text { Minimize: } \sum_{s \in S} c_{s} x_{s}+\sum_{i \in I}\left(f_{i} y_{i}+F_{i} Y_{i}+g_{i} z_{i}\right)\{\text { Total costs }\} \tag{3.1}
\end{equation*}
$$

Subject to:

$$
\begin{align*}
& \left.\sum_{s \in S} d_{s i} x_{s}=b_{i}+y_{i}+Y_{i}-z_{i}, \text { for each } i \in I \text { \{Total demand }\right\}  \tag{3.2}\\
& \left.\sum_{s \in S} a_{s j} x_{s}=1 \text {, for each } j \in J \text { \{Employee gets } 1 \text { schedule }\right\}  \tag{3.3}\\
& \left.\sum_{j \in J} w_{i j}^{q} \geq b_{i}^{q} \text {, for each } q \in Q \text {, and } i \in I \text { \{Qualification demand }\right\}  \tag{3.4}\\
& \left.\sum_{q \in Q} w_{i j}^{q}=\sum_{s \in S} d_{s i} a_{s j} x_{s}, \text { for each } i \in I, \text { and } j \in J \text { \{Task assignment }\right\}  \tag{3.5}\\
& w_{i j}^{q}-h_{j}^{q} \leq 0, \text { for each } q \in Q, i \in I, \text { and } j \in J\{\text { Qualified for task }\} \tag{3.6}
\end{align*}
$$

$x_{s} \in\{0,1\}$, for each $s \in S$ \{No schedule partition $\}$
$w_{i j}^{q} \in\{0,1\}$, for each $q \in Q, i \in I$, and $j \in J$ \{No task partition $\}$
$0 \leq y_{i} \leq o_{i}$, for each $i \in I$ \{Allowed surplus\}

Note that the notation has been slightly changed. The objective function and constraints (3.2), (3.3), (3.7) and (3.11) are (approximately) the same as before. Constraint (3.9) and (3.10) are a splitting, such that an upper bound on the allowed surplus in a shift can be modeled.

Constraint (3.4) ensures that each 'task' is performed by enough personnel. Constraints (3.5) and (3.8) together state that an employee is assigned to exactly one task for a shift if and only if he is assigned to work in that shift. Finally, constraint (3.6) ensures that an employee is qualified for the assigned task. This last constraint can also be preprocessed: when employee $j$ does not have qualification $q$, set $w_{i j}^{q}=0$ in advance for each $i \in I$, and $j \in J$.

This formulation has two major disadvantages: it easily leads to symmetry issues and the number of constraints is large.
For the symmetry argument, consider employees 1 and 2 with the same set of 2 qualifications. When there exists a solution in which employee 1 is assigned to task $\alpha$ and employee 2 to task $\beta$, a solution in which the assignment is the other way around is also possible. However, the meaning of the solution is the same: the qualification demands are fulfilled with the same group of employees.
It might be possible to add constraints and cost parameters stating that whenever employees 1 and 2 are both available for qualification $q$ in shift $i$, employee 1 is taken. However, as stated by the second argument, the number of constraints in the solution is already quite large without these new constraints. The reason for this is that the constraints (3.4), (3.5) and (3.6) are designed to hold for every combination of values from two or even three domains. Therefore, the number of constraints is cubic in the number of employees, shifts and qualifications.

Hence we will investigate whether a different approach to express the qualification constraints, without symmetry issues, is possible.

### 3.3 Introducing qualification groups

In this section, we will introduce the concept of qualification groups: a general solution approach to overcome the symmetry issues.

Definition 3.1 (Qualification group). A set of employees whose qualifications are exactly the same.

Suppose, we have the following set of possible qualifications: $Q=\{1,2,3\}$. We can define the set with all possible combinations of qualifications as the powerset of $Q$ without the empty set ${ }^{2}$. Hence the set $Q^{*}$ of qualification groups is defined as:

$$
\begin{equation*}
Q^{*}=\mathcal{P}(Q) \backslash\{\emptyset\} \tag{3.12}
\end{equation*}
$$

In this example, this means:

$$
\begin{array}{rlrrrrrr}
Q^{*} & =\left\{\begin{array}{rrrrrr}
\{1\}, & \{2\}, & \{3\}, & \{1,2\}, & \{1,3\}, & \{2,3\},
\end{array}\{1,2,3\}\right\} \\
& =\left\{\begin{array}{rrrr} 
& Q^{1}, & Q^{2}, & Q^{3},
\end{array} Q^{12},\right. & Q^{13}, & Q^{23}, & \left.Q^{123}\right\} \tag{3.13}
\end{array}
$$

An employee with qualifications 1 and 3 , who is allowed to perform 'task' 1 or 'task' 3 , is only a member of qualification group $Q^{13}$. He is thus not also a member of the groups $Q^{1}$ and $Q^{3}$.

For a particular shift $i$, let $b_{i}=\left[b_{i}^{1}, b_{i}^{2}, b_{i}^{3}\right]$ be the minimal occupancy demands per qualification for shift $i$. Let $Q_{i}^{q} \subseteq Q^{q}$ be the employees of qualification group $Q^{q}$ assigned to shift $i$. Then, it is not sufficient merely to check the number of employees assigned to each 'task':

$$
\begin{align*}
& \left|Q_{i}^{1}\right|+\left|Q_{i}^{12}\right|+\left|Q_{i}^{13}\right|+\left|Q_{i}^{123}\right| \geq b_{i}^{1} \\
& \left|Q_{i}^{2}\right|+\left|Q_{i}^{12}\right|+\left|Q_{i}^{23}\right|+\left|Q_{i}^{123}\right| \geq b_{i}^{2}  \tag{3.14}\\
& \left|Q_{i}^{3}\right|+\left|Q_{i}^{13}\right|+\left|Q_{i}^{23}\right|+\left|Q_{i}^{123}\right| \geq b_{i}^{3}
\end{align*}
$$

[^11]Proof. Take the demand $b_{i}=[1,1,1]$ and qualification group sizes $\left|Q_{i}^{12}\right|=1$ and $\left|Q_{i}^{23}\right|=1$. The constraints are satisfied, but only 2 employees are assigned to perform 3 tasks.

It is possible to add a constraint, that states that the total number of employees divided over all tasks should be correct. Unfortunately, this does not solve the problem:

$$
\begin{equation*}
\left|Q_{i}^{1}\right|+\left|Q_{i}^{2}\right|+\left|Q_{i}^{3}\right|+\left|Q_{i}^{12}\right|+\left|Q_{i}^{13}\right|+\left|Q_{i}^{23}\right|+\left|Q_{i}^{123}\right| \geq b_{i}^{1}+b_{i}^{2}+b_{i}^{3} \tag{3.15}
\end{equation*}
$$

Proof. Take the demand $b_{i}=[2,4,2]$ and qualification group sizes $\left|Q_{i}^{12}\right|=2,\left|Q_{i}^{23}\right|=2$ and $\left|Q_{i}^{3}\right|=4$. Although there are 8 people assigned, 'task' 1 and 2 are only spread over 4 employees. However, there are actually 6 employees needed to perform these two tasks.

The last sentence of the proof entails the solution for the problem. Besides the above mentioned four constraints, it is also needed to add constraints for all combinations of tasks to guarantee there is enough staff assigned to perform the combination of tasks. Hence we also need the following constraints:

$$
\begin{align*}
& \left|Q_{i}^{1}\right|+\left|Q_{i}^{2}\right|+\left|Q_{i}^{12}\right|+\left|Q_{i}^{13}\right|+\left|Q_{i}^{23}\right|+\left|Q_{i}^{123}\right| \geq b_{i}^{1}+b_{i}^{2} \\
& \left|Q_{i}^{1}\right|+\left|Q_{i}^{3}\right|+\left|Q_{i}^{12}\right|+\left|Q_{i}^{13}\right|+\left|Q_{i}^{23}\right|+\left|Q_{i}^{123}\right| \geq b_{i}^{1}+b_{i}^{3}  \tag{3.16}\\
& \left|Q_{i}^{2}\right|+\left|Q_{i}^{3}\right|+\left|Q_{i}^{12}\right|+\left|Q_{i}^{13}\right|+\left|Q_{i}^{23}\right|+\left|Q_{i}^{123}\right| \geq b_{i}^{2}+b_{i}^{3}
\end{align*}
$$

To conclude: the number of required constraints is exponential in the number of possible qualifications (there are exactly $2^{|Q|}-1$ of such constraints needed). Since these constraints must be added for every possible shift, the number of constraints in the model will explode. However, when the number of possible qualifications is low, it is workable to use this construction.

### 3.4 The UMC master problem

Recall the occupancy demands for the UMC case for a day shift as stated in section 1.6:
Theorem 3.1 (Day shift occupancy demand). At least 5 employees with an MC-qualification and 4 more with at least a VK-qualification should be assigned to this shift. At least 2 out of these nine should have an SVK-qualification too. In total, there should be at least 11 employees working in this shift.

Since students can be treated as 'having no qualification at all', we should take the following 3 types of qualifications into account while modeling our main problem:

- SVK: Senior Nurse
- MC: Medium Care Nurse
- VK: Nurse

In this way, we need to use $2^{3}-1=7$ constraints for each shift. Luckily, we can exploit the overlapping of qualifications. As stated in chapter 1, an employee with the SVK- or MC-qualification always has the basic VK-qualification too. Therefore, the following qualification groups are already empty and therefore not needed in our constraints: $Q^{s v k}, Q^{m c}, Q^{s v k, m c}$.
Furthermore, it is only needed that of the selected MC-nurses and VK-nurses, enough people have an SVK-qualification, since the SVK-demand is not defined as a kind of single 'task'. Hence, we only need one simple constraint for the SVK-demand. An assigned nurse, who counts as an MCor VK-nurse, also counts in the SVK-constraint when the nurse has this qualification.

Let the expression $b_{i}^{m c \vee v k}$ be defined as the demand of all employees having an MC-qualification (and hence also a VK-qualification) or only a VK-qualification. Actually 3 constraints are needed
for a particular shift $i$ :

$$
\begin{array}{lll}
\left|Q_{i}^{s v k, m c, v k}\right|+\left|Q_{i}^{s v k, v k}\right| & & \geq b_{i}^{s v k} \\
\left|Q_{i}^{s v k, m c, v k}\right| & +\left|Q_{i}^{m c, v k}\right| & \geq b_{i}^{m c}  \tag{3.17}\\
\left|Q_{i}^{s v k, m c, v k}\right|+\left|Q_{i}^{s v k, v k}\right| & +\left|Q_{i}^{m c, v k}\right|+\left|Q_{i}^{v k}\right| & \geq b_{i}^{m c v v k}
\end{array}
$$

The first constraint ensures that there are enough employees with an SVK-qualification. The second ensures that enough employees can perform the MC-'task'. The last one ensures that enough employees can perform the VK-'task', while not already being counted for the MC-task.
Note that there is no such thing as a minimum VK-qualification demand. With $\left(b_{i}^{m c \vee v k}\right)$, it is only made sure that the desired demand of employees with a VK- (and optionally an MC-) qualification is met.

Example 3.1. Consider the day shift requirements as defined in section 1.6. For a day shift $i$, we have the following values for the variables $b_{i}^{q}:\left\{b_{i}^{s v k}=2, b_{i}^{m c}=5, b_{i}^{m c \vee v k}=9\right\} .{ }^{3}$
Suppose 5 employees with an MC- (hence also a VK-) qualification are assigned, 1 of them also has an SVK-qualification. Furthermore, 3 assigned employees only have a VK-qualification. Finally, 1 assigned employee has both a VK-qualification and an SVK-qualification.

This leads to the following system of equations:

$$
\begin{array}{lll}
1+1 & & \geq 2 \\
1 & +4 & \geq 5  \tag{3.18}\\
1+1 & +4+3 & \geq 9
\end{array}
$$

Indeed the occupancy demand is satisfied for every qualification. An additional rule will be added to satisfy the total employee demand $\left(b_{i}=11\right)$.
The constraints in equation (3.17) can be simplified even more. Observe that the three individual constraints concern all employees with the SVK-qualification, all employees with the MCqualification and all employees with the VK-qualification respectively. We can use the following notation (according to the legend in section 3.1):
$h_{j}^{\mathbf{v k}}$ Whether employee $j$ has the VK-qualification (all employees except students).

$$
h_{j}^{\mathrm{vk}}= \begin{cases}1 & \text { when employee } j \text { has the VK-qualification. } \\ 0 & \text { otherwise. }\end{cases}
$$

$h_{j}^{\mathrm{mc}}$ Whether employee $j$ has the MC-qualification.

$$
h_{j}^{\mathrm{mc}}= \begin{cases}1 & \text { when employee } j \text { has the MC-qualification. } \\ 0 & \text { otherwise. }\end{cases}
$$

$h_{j}^{\text {svk }}$ Whether employee $j$ has the SVK-qualification.

$$
h_{j}^{\text {svk }}= \begin{cases}1 & \text { when employee } j \text { has the SVK-qualification. } \\ 0 & \text { otherwise. }\end{cases}
$$

Let $J$ denote the set of all employees and let $\chi_{i j}$ denote whether employee $j$ is assigned to shift $i$.

[^12]The final system of qualification equations for a shift $i$ will then be:

$$
\begin{align*}
& \sum_{j \in J} h_{j}^{\mathrm{svk}} \chi_{i j} \geq b_{i}^{s v k} \\
& \sum_{j \in J} h_{j}^{\mathrm{mc}} \chi_{i j} \geq b_{i}^{m c}  \tag{3.19}\\
& \sum_{j \in J} h_{j}^{\mathrm{vk}} \chi_{i j} \geq b_{i}^{m c \vee v k}
\end{align*}
$$

Employee surplus for a particular qualification is not a problem, as long as the total number of employees that needs to be active in the shift is not too high. However, employee shortage for a particular qualification is a problem. Again, it is possible to force the lower bound and upper bound of the qualification occupancy demand (instead of penalizing it), but then it remains difficult to find an allowed initial solution in the column generation phase. Also, in case the computed solution includes a shortage somewhere, it is likely that it can be easily solved by the scheduling employee.

Hence, besides the total shortage penalty, we will introduce a shortage penalty per qualification: $z_{i}^{\mathbf{s v k}}$ The number of SVK shortage in shift $i$.
$z_{i}^{\mathrm{mc}}$ The number of MC shortage in shift $i$.
$z_{i}^{\mathbf{v k}}$ The number of VK shortage in shift $i$.

### 3.4.1 Objective function

The objective function now becomes a weighted function of employee satisfaction and surplus / shortness costs ${ }^{4}$ :

$$
\begin{equation*}
\text { Minimize: } \sum_{s \in S} c_{s} x_{s}+\sum_{i \in I}\left(f_{i} y_{i}+F_{i} Y_{i}+g_{i}\left(z_{i}^{\mathrm{svk}}+z_{i}^{\mathrm{mc}}+z_{i}^{\mathrm{vk}}+z_{i}\right)\right)\{\text { Total costs }\} \tag{3.20}
\end{equation*}
$$

### 3.4.2 Constraints

Most constraints are about the same as defined in sections 2.3 and 3.2. Only the specific task assignment constraints have been replaced by the aforementioned demand constraint types (3.17):

$$
\begin{align*}
& \left.\sum_{s \in S} d_{s i} x_{s}=b_{i}+y_{i}+Y_{i}-z_{i}^{\mathrm{svk}}-z_{i}^{\mathrm{mc}}-z_{i}^{\mathrm{vk}}-z_{i}, \text { for each } i \in I \text { \{Total demand }\right\}  \tag{3.21}\\
& \sum_{s \in S} h_{j}^{\mathrm{svk}} d_{s i} a_{s j} x_{s} \geq b_{i}^{\mathrm{svk}}-z_{i}^{\mathrm{sk}}, \text { for each } i \in I\{\text { SVK demand }\}  \tag{3.22}\\
& \sum_{s \in S} h_{j}^{\mathrm{mc}} d_{s i} a_{s j} x_{s} \geq b_{i}^{\mathrm{mc}}-z_{i}^{\mathrm{mc}}, \text { for each } i \in I\{\mathrm{MC} \text { demand }\}  \tag{3.23}\\
& \left.\sum_{s \in S} h_{j}^{\mathrm{vk}} d_{s i} a_{s j} x_{s} \geq b_{i}^{\mathrm{mcvvk}}-z_{i}^{\mathrm{mc}}-z_{i}^{\mathrm{vk}}, \text { for each } i \in I \text { \{addable VK demand }\right\}  \tag{3.24}\\
& \sum_{s \in S} a_{s j} x_{s}=1, \text { for each } j \in J\{\text { Employee gets } 1 \text { schedule }\} \tag{3.25}
\end{align*}
$$

[^13]\[

$$
\begin{align*}
& \left.x_{s} \in\{0,1\}, \text { for each } s \in S \text { \{No schedule partition }\right\}  \tag{3.26}\\
& \left.0 \leq y_{i} \leq o_{i}, \text { for each } i \in I \text { \{Allowed surplus }\right\}  \tag{3.27}\\
& \left.Y_{i} \geq 0, \text { for each } i \in I \text { \{Count too much surplus }\right\}  \tag{3.28}\\
& \left.z_{i}^{\text {svk }}, z_{i}^{\text {mc }}, z_{i}^{\text {vk }}, z_{i} \geq 0, \text { for each } i \in I \text { \{Count shortages }\right\} \tag{3.29}
\end{align*}
$$
\]

Constraints (3.22), (3.23) and (3.24) try to force the qualification demands. When there is a shortness for a specific qualification, a penalty will be added to the objective value. This is done using decision variables $z_{i}^{q}$. According to the constraint formulation, an employee counts for a qualification demand in a shift if (1) he has the qualification and (2) he is assigned to a roster that is made for him in which the shift has been assigned. Constraint (3.21) is used to set the decision variable $z_{i}$, to satisfy the total employee demand including unqualified employees (students). All these decision variables can not be negative (constraint (3.29)).
The problem formulation is rather handy, since no constraints are squared for combinations of domains (e.g. formulations like 'for each $i \in I$ and $j \in J$ '). The total number of constraints is only $4 m+n$, in other words: linear in the number of shifts and the number of employees.

It is possible to add an extra constraint to restrict the number of students assigned to a shift, but we did not use that.

### 3.5 Solving the ILP

Using the approach as will be described in chapter 4, a set of possible rosters for all employees is created. It is impossible to add all those rosters to the problem. Hence, revised simplex or column generation (see section 2.2.1) is necessary to keep the problem manageable.

First, we perform an LP-relaxation. The integrality constraint (3.26) is changed in $x_{s} \geq 0$. Note that the upper bound of 1 is still forced by constraint (3.25), since $x_{s}$ can not be negative.
Constraint (3.25) gives dual multipliers $\lambda_{1}, \ldots, \lambda_{n}$.
The demand constraints (3.21), (3.22), (3.23) and (3.24) give the dual multipliers $\pi_{1}, \ldots, \pi_{m}$; $\pi_{1}^{\mathrm{svk}}, \ldots, \pi_{m}^{\mathrm{svk}} ; \pi_{1}^{\mathrm{mc}}, \ldots, \pi_{m}^{\mathrm{mc}}$ and $\pi_{1}^{\mathrm{vk}}, \ldots, \pi_{m}^{\mathrm{vk}}$ respectively.
The reduced cost of a variable $x_{s}$ is thus given by:

$$
\begin{align*}
c_{s}^{\prime} & =c_{s}-\lambda_{j} a_{s j}-\sum_{i=1}^{m}\left(\pi_{i} d_{s i}+\pi_{i}^{\mathrm{svk}} q_{j}^{\mathrm{svk}} d_{s i} a_{s j}+\pi^{\mathrm{mc}} q_{j}^{\mathrm{mc}} d_{s i} a_{s j}+\pi^{\mathrm{vk}} q_{j}^{\mathrm{vk}} d_{s i} a_{s j}\right) \\
& =c_{s}-\lambda_{j} a_{s j}-\sum_{i=1}^{m} d_{s i}\left[\pi_{i}+a_{s j}\left(\pi_{i}^{\mathrm{svk}} q_{j}^{\mathrm{svk}}+\pi^{\mathrm{mc}} q_{j}^{\mathrm{mc}}+\pi^{\mathrm{vk}} q_{j}^{\mathrm{vk}}\right)\right] \tag{3.30}
\end{align*}
$$

Since the problem of adding good rosters will be individually solved for every employee, the term $\lambda_{j} a_{s j}$ can be 'ignored'. Given a set $S_{j}$ of valid rosters for an employee $j$, the parameter $a_{s j}$ is always 1 and can also be 'ignored'. The values of $q_{j}^{\text {svk }}, q_{j}^{\mathrm{mc}}$ and $q_{j}^{\mathrm{vk}}$ are known in advance and are thus constants. Introducing a selection variable $x_{s}$, the individual pricing problem thus becomes:

$$
\begin{equation*}
\text { Minimize: } \sum_{s \in S_{j}}\left(c_{s}-\sum_{i=1}^{m} d_{s i}\left[\pi_{i}+\pi_{i}^{\mathrm{svk}} q_{j}^{\mathrm{svk}}+\pi^{\mathrm{mc}} q_{j}^{\mathrm{mc}}+\pi^{\mathrm{vk}} q_{j}^{\mathrm{vk}}\right]\right) x_{s} \tag{3.31}
\end{equation*}
$$

Subject to:

$$
\begin{align*}
& \sum_{s \in S_{j}} x_{s}=1\{\text { Employee gets } 1 \text { schedule }\}  \tag{3.32}\\
& x_{s} \in\{0,1\}\{\text { No schedule partition }\} \tag{3.33}
\end{align*}
$$

The following solution steps will be performed:

1. For every individual employee, his $\theta$ rosters with lowest costs will be added to the problem as a start. If an employee has less than $\theta$ possible rosters, all his rosters will be added to the master problem. With this selection, an initial solution will be calculated by solving the LP.
2. In each iteration, the $\gamma$ best rosters (the ones with lowest negative reduced cost) are added to the problem for every employee. It is possible to take $\gamma=1$, but since a lot of individual rosters are approximately the same, it is more efficient to take a larger amount of rosters with negative costs. It is likely that, in later iterations, most of these rosters would have been added anyway.

Besides, this observation is also mentioned by He and Qu (2012)[14]: "Choosing the optimal column (with the minimum negative cost) is not always the best choice. In practice, any feasible columns which have negative reduced cost can serve as candidates to enter the master problem."
3. When there is no employee for whom a roster with negative reduced costs can be added, the LP has been solved to optimality and the iterative process stops. The process is also aborted when a predefined time limit has been reached.
4. Finally, the solution of the ILP is calculated using all the columns that were added to the LP-relaxation. To try to obtain a better bound, the $\phi$ most-preferred rosters based on the dual multipliers of the optimal solution of the LP-relaxation are also added to the problem. The process is aborted when the optimal value has been calculated or a predefined time limit has been reached (in case the ILP-solver has problems proving the optimality).

## Chapter 4

## Creation of individual rosters

The previous chapter discussed how a complete schedule for employees can be created, using a set of possible schedules for each individual employee. In this chapter, the approach to create individual schedules for a roster period of 6 weeks will be discussed.
On a particular day of the roster, 4 different shifts can be assigned (see chapter 1). Although not all patterns of shifts are permitted, the number of possible valid shift combinations for 6 weeks (and even 3 or 4 weeks) is already tremendous. This implies that additional roster bounding should be applied. Examples of bounding rules are:

- Do not allow shift blocks (see section 1.3) with a shorter or longer length than a predefined parameter (which can differ for various shift types).
- Only allow a small amount of shift block patterns (e.g. do not allow Day $\rightarrow$ Late, but only Day $\rightarrow$ Free $\rightarrow$ Late).
- Force a weekend off after a few weeks having no free weekend at all.

We will define such rules in this chapter.
Section 4.1 discusses a main generation system, which is able to create rosters for most employees for a limited number of weeks. In section 4.2, we will discuss the mathematical problem of finding rosters for a period of 6 weeks. Some solution methods will be proposed, each with its own limitations, and a final system will be introduced. Thereafter, section 4.3 will introduce the possible soft requirements and cost functions being used. Finally, section 4.4 will treat special generation systems for employees with a low amount of possible rosters or special regulations.

### 4.1 Depth-first search with forwards rotation

The most natural way of generating rosters is to build them from the start till the end. First, the data for a particular roster period must be inserted: demands and preferences for (parts of) days off, shift preferences, shift block preferences, etc. Thereafter, a parameter configuration that is suitable to build rosters for the employees should be selected. This configuration is used to prevent the generation of 'bad' rosters and to try to obtain a manageable number of rosters. Manageable means that the number of rosters is such that all rosters can easily fit into computer memory. Naturally, the parameter configuration must ensure that every employee obtains at least a few possible rosters, to have some choice.
Section 4.1.1 will discuss the main generation algorithm, whearas sections 4.1.2 and 4.1.3 will discuss the possible parameter settings.

### 4.1.1 The algorithm

Depth-first search (DFS) is a basic algorithm for traversing or searching a graph. Starting from a specific node $s$ of a graph (e.g. the root node of a tree), the graph is explored until a node $v$ without outgoing edges to unexplored neighbour-nodes is reached. Hence, a simple path $(s, v)$ is created.

Then, the algorithm performs some backtracking steps (following the path in reverse order, thus starting at $v$ ) until a pre-visited node which has still outgoing edges to unexplored nodes is reached. From there, the algorithm continues as just described. When backtracking does not find a new unexplored node, then the algorithm stops (or a still unvisited node is selected as a new root node).

Algorithm 1 (modified from Cormen et al. (2009)[7]) shows a global sketch for DFS on a graph $G$ from a single start node $s$.

```
Algorithm 1 DEPTH-FIRST SEARCH
    \(\triangleright\) Initialize algorithm and explore graph \(G\) from root node \(s\)
    function \(\operatorname{DFS}(G, s)\)
        for all \(u \in V[G]\) do
            visited \([u] \leftarrow\) False
        end for
        DFS-Visit \((s)\)
    end function
    function DFS-VISIT( \(u\) )
        visited \([u] \leftarrow\) True
        for all \(v \in N\) eighbours \((u)\) do
            if visited \([v]=\) False then
            \(\operatorname{DFS}-\operatorname{VISIT}(v)\)
        end if
        end for
    end function
```

Creating a roster for $n$ days in a depth-first manner could be considered as exploring an (implicit) tree with a variant of depth-first search. The root of the tree represents an undecided shift allocation for the first day of the roster (for convenience, we will call the first day of the roster 'day 0 ' and the last day 'day $n-1$ '). The node has four edges to child nodes, each one representing a different allocation (Day, Late, Night, Free) for day 0. Each of these child nodes subsequently represents an undecided shift allocation for the second day ('day 1 '), while having specified the shift for the first day. In general: a node for a day $d$ represents a decision point for the shift allocation for day $d$, while having allocated the shift for days $\mathcal{D}=\left\{d^{\prime} \mid 0 \leq d^{\prime}<d\right\}$. The nodes in layer $n$ represent unique full schedules (since there is a unique path to them). See Figure 4.1.

Since for every day 4 possible shifts can be assigned, the total number of possible rosters for an employee is at most $4^{n}$, which is already $7.2 \times 10^{16}$ for only four weeks ( 28 days). Furthermore, the graph in Figure 4.1 has 1 node in layer 0,4 nodes in layer 1, 16 nodes in layer 2, etc. Hence, the total number of nodes in this graph amounts to $\sum_{d=0}^{n} 4^{d}$.

It is naturally impossible to store these numbers of rosters or nodes into the memory of a modern computer. Fortunately, most of the rosters in the graph are not even close to being meaningful and for many rosters it is already possible to determine this after performing only a few decision steps in the tree.

Hence, we are able to create a DFS-based algorithm that gradually creates a roster tree, which is cut off when it is certain that the assignments at the start of the roster will not lead to a meaningful (or good) schedule. We will call this bounded DFS (see Algorithm 2).
The algorithm works as follows: it is initialized (DFS-CrEATE-ROSTERS $(n)$ ) by creating an empty schedule that is suitable for holding shifts for $n$ days. Then, it will firstly try to add a 'Day' shift


Figure 4.1: Implicit graph representation for creating a roster in a depth-first manner.
for the first day of the roster. If this shift is allowed, a 'Day' shift for the second day of the roster will be considered, etc. If a shift on a certain day $d$, given the already allocated shifts on days $0 \ldots d-1$, appears to be illegal by introducing the violation of a bounding rule (method Check(roster)), then all rosters which start with the indicated shifts are not allowed, so the branch can be cut off and another shift (for instance a 'Late' shift) can be tried for day $d$. If there is no possible shift left for day $d$ (given the set of previous shifts), the algorithm will backtrack to the last encountered promising node.

Definition 4.1 (Promising node). A decision node for day d for which not all possible shifts have been considered to allocate on day d. Hence it is still possible to create unexplored branches starting from this node.
Because of the backtracking, the algorithm will not only consider rosters which start with a 'Day' shift, but also rosters starting with a 'Late' or 'Night' shift or a day off.

When a node in layer $n$ has been reached, a valid schedule for the employee is found and added to a set. The algorithm will backtrack to the last encountered promising node. When there is no such node available, the algorithm finishes with a return of the set of valid rosters.

In this way, exactly all possible schedules which satisfy the predefined parameter configuration will be generated. Furthermore, all generated schedules are different. It is impossible to have duplicated schedules for a single employee.

Finally, a nice benefit of the system is that time does not play a big role. The generation of rosters for a period only has to be done once. Hence this process can be performed at 'any' suitable time. However, the size of the active memory of the computer does play a role, since all rosters should be allocated quickly in the linear programming phase (chapter 3) and thus have to be stored in RAM memory.

### 4.1.2 Basic parameters and bounding rules

The check-method uses some bounding rules to early prevent the generation of unsatisfiable rosters (according to the rules in chapter 1). This subsection will provide an overview of the main rules. Since rosters are generated for a single employee at a time, all sets and parameters will not have an employee parameter to simplify the notation.

An employee has variables $\mathcal{S}_{m}[d]$ and $\mathcal{S}_{f}[d](d \in\{0, \ldots, n-1\})$. A variable $\mathcal{S}_{m}[d]$ contains at most 1 shift, that must be allocated on day $d$ for a specific employee. A variable $\mathcal{S}_{f}[d]$ contains a

```
Algorithm 2 Bounded Depth-First search rostering
    \(\triangleright\) Possible shifts on a day and a set to store valid rosters into
    pos_shifts \(\leftarrow\{\) Day, Late, Night, Free \(\}\)
    valids \(\leftarrow\) NEW Set()
                                    \(\triangleright\) Initialize algorithm and start building from empty schedule
    function DFS-Create-Rosters(n)
        \(r \leftarrow\) NEW Roster[n]
        \(\operatorname{AddShiftOnDAy}(r, 0, n)\)
        return valids
    end function
                            \(\triangleright\) Extend the roster by recursively adding shifts
    function AddShiftOnDAy (roster, day, \(n\) )
        if \(d a y=n\) then
            valids.ADD (roster)
            return
        end if
        for all shift \(\in\) pos_shifts do
            if Check(roster, day, shift) then
                roster \([\) day \(] \leftarrow\) shift
                AddShiftOnDay (roster, \(d a y+1, n\) )
            end if
        end for
    end function
                                    \(\triangleright\) Check whether it is valid to add a shift
    function CHECK (roster)
        if Roster violates a bounding rule then
            return FALSE
        else
            return TRUE
        end if
    end function
```

list of at most 3 shifts (since there are 4 shifts possible and one of them must be inserted) which are forbidden to allocate on day $d$ for a specific employee. We prevent impossible combinations of these variables in advance. Hence IF $\mathcal{S}_{m}[d] \neq \emptyset$ then $\mathcal{S}_{m}[d] \cap \mathcal{S}_{f}[d]=\emptyset$.

Furthermore, for a roster period of $n$ days ( $k$ weeks of 7 days ${ }^{1}$, each week is specified as a period Monday - Sunday), an employee has parameters:

- $a$, the number of workdays during a normal week (defined as Monday - Sunday). This is the size of the appointment of the employee.
- $a_{k}$, the number of workdays in $k$ weeks.
- $a^{+}$, the allowed number of workdays surplus in a week (normally 1). Allocating an extra workday above the amount $a$, means allocating less workdays in a later week.
- $a_{k}^{+/-}$, the allowed number of workdays surplus or less in $k$ weeks (normally 1). Allocating an extra workday or one short, might be done to help getting the occupancy right or to help creating rosters according to a specific pattern. It is possible to split this parameter in a in 'surplus' and 'shortness' variant, but this was not needed for the problem statement.
- $\mathcal{W}_{\text {min }}$ and $\mathcal{W}_{\text {max }}$, which contain the minimum and maximum number of free weekends during $k$ weeks.
- $\mathcal{W}_{\text {row }}$, which contains the maximum number of work weekends in a row during $k$ weeks.
- $c$, which contains the maximum number of consecutive workdays.

[^14]Let $\omega(d)$ be a function which maps a day $d$ to a weekday $\in\{$ Monday, $\ldots$, Sunday $\}$. In the situations below, the algorithm will stop exploring a node any further, due to infringing a hard constraint. Hence, the sketched situations are forbidden.

### 4.1.2.1 Hard shift assignment constraints

- Allocation of a (forbidden) shift $s \in \mathcal{S}_{f}[d]$ on a day $d$.
- Allocation of a shift $s \neq \mathcal{S}_{m}[d]$ on a day $d$ when $\mathcal{S}_{m}[d] \neq \emptyset$.


### 4.1.2.2 Weekend shifts

- Let $f$ be the number of allocated free weekends before day $d$. Allocation of a free shift on a $\omega(d)=$ Saturday, when $f=\mathcal{W}_{\text {max }}$ (too much free weekends) or allocation of a work shift on $\omega(d)=$ Saturday, when $\left\lfloor\frac{n-d}{7}\right\rfloor+f<\mathcal{W}_{\text {min }}$ (impossible to obtain the minimum number of free weekends).
- Allocation of a shift on a $\omega(d)=$ Sunday that differs from the shift on day $d-1$ (Saturday).
- Let $g$ be the number of work weekends since the last free weekend. Allocation of a work shift on a day $\omega(d)=$ Saturday, when $g=\mathcal{W}_{\text {row }}$.


### 4.1.2.3 Workdays restrictions

Let $a^{*}$ be the total number of assigned workdays in the roster.

- Allocation of a work shift on a day $d$, when there are no free shifts in days $d-1, \ldots, d-c$ (too much workdays after each other).
- Allocation of a work shift on a day $d$ during a week $w$, when the number of scheduled workdays in week $w$ is already equal to $a+a^{+}$.
- Allocation of a work shift on a day $d$, when $a^{*} \geq a_{k}+a_{k}^{+/-}$(workdays overflow).
- Allocation of a free shift on a day $d$, when $a_{k}-a_{k}^{+/-}-a^{*} \leq n-(d+1)$ (impossible to obtain the minimal amount of workdays in the remaining days).
- Allocation of a free shift on a day $d$, when day $d-1$ contains a work shift, day $d-2$ contains a free shift and a single workday between two free shifts is not allowed.
- Let $f$ be the minimum length of a block of days off. Allocation of a work shift on a day $d>d^{\prime}$, when $d^{\prime}$ contains a work shift, days $\mathcal{D}=\left\{d^{\prime \prime} \mid d^{\prime}<d^{\prime \prime}<d\right\}$ each contain a free shift, but $|D|<f$ (minimum length of a block of free shifts).


### 4.1.2.4 Night shift restrictions

Night shifts can only occur on Monday and Tuesday, Wednesday and Thursday, Monday until Thursday and Friday until Sunday.

- Allocation of a night shift on a day $\omega(d) \in\{$ Tuesday, Thursday, Saturday, Sunday $\}$, when the shift on day $d-1$ is not a night shift.
- Allocation of a night shift on a day $\omega(d) \in\{$ Monday, Friday $\}$, when the shift on day $d-1$ is also a night shift.
- No allocation of a night shift on a day $\omega(d) \in\{$ Tuesday, Thursday, Saturday, Sunday $\}$, when the shift on day $d-1$ is a night shift.
- Let $g$ be the number of night shifts currently scheduled in the roster and $g^{\max }$ be the maximum number of night shifts allowed in he roster. Allocation of a night shift on a day $d$, when $g=g^{\max }$.

Note: The rule of at least two free days after a night shift will be specified later.

### 4.1.3 Parameters to stimulate a forwards rotating roster

Some additional parameters are introduced to force a forwards-rotating roster (see section 1.3). It is created by using a scheme or pattern. The pattern guarantees that shift types (Day/Late/Night) are regularly alternated and the employee has shift blocks (he works a few consecutive days in the same shift). By varying the possible block lengths, it is easier to grant a hard shift requirement (e.g. a (regular) day off).

An example scheme could be the following. The employee's schedule has the following shifts in order:

1. At least 2 and at most 6 day shifts.
2. Optionally some days off.
3. At least 1 and at most 3 late shifts.
4. Optionally some days off.
5. Optionally 2-4 night shifts.
6. At least 2 days off.

To forbid an occurrence of a day shift directly succeeding a late shift, it can be demanded that an employee has at least 1 day off before he works a day shift. The idea behind the previous scheme is the fact that more employees are needed during a day shift than during a late shift. Hence, an employee will retrieve more day than late shifts in his schedule and thus there is a bigger chance that:

1. All day shifts can be filled.
2. There is only little surplus during the late shifts.

Furthermore, an employee has only $0-2$ blocks of night shifts during a period of six weeks. Hence the night shift block in the pattern is optional.
The following additional bounding rules (hence, the sketched situations are forbidden) are introduced:

- Allocation of a workshift $w$ on day $d$, when the current length of the $w$-shift block is already maximal.
- Allocation of a shift $s$ on day $d$, when workshift $w^{\prime}(\neq s)$ was performed on day $d-1$ and the current length of the $w^{\prime}$-shift block is less than the defined minimum length of a block of $w^{\prime}$-shifts.
- Allocation of a workshift $w$ on day $d$, when there is a workshift $w^{\prime}(\neq w)$ on a day $d^{\prime}(<d)$ and the number of days off between days $d^{\prime}$ and $d$ is less than the minimum amount of days off after a $w^{\prime}$-shift block.
- Allocation of a workshift $w$ on day $d$, when there is a workshift $w^{\prime}(\neq w)$ on a day $d^{\prime}(<d)$ and the number of days off between days $d^{\prime}$ and $d$ is less than the minimum amount of days off before a $w$-shift block.
- Allocation of a workshift $w$ on day $d$, when there is a workshift $w^{\prime}(\neq w)$ on a day $d^{\prime}(<d)$ and the number of days off between days $d^{\prime}$ and $d$ is more than the maximum amount of days off before a $w$-shift block.
- Allocation of a free shift $f$ on day $d$, when there is a workshift $w^{\prime}$ on a day $d^{\prime}(<d)$ and the number of days off between days $d^{\prime}$ and $d$ is at least the maximum amount of days off after a $w^{\prime}$-shift block.

To prevent an unnecessary restriction of the number of possible rosters for an employee, we will not check all these additional rules when the algorithm is at the start of the roster.

Definition 4.2 (Start of roster). The algorithm is at the start of the roster when either:

1. The algorithm selects a shift for day 0 .
2. Let $s_{0}$ be the shift assigned on day 0 . The algorithm selects a shift for day $d>0$ and all shifts on days $0 \ldots d-1$ are equal to $s_{0}$.

In other words: we allow the roster to start somewhere in the designated pattern. The following example tries to clarify this.

Example 4.1 (Relaxed start of roster). Consider the example scheme mentioned hereinbefore. The scheme demands that every block of day shifts consists of at least 2 consecutive days, but we will allow a single day shift at the start of the roster. This is the only place in the roster where a single day shift could occur.

In order to make this possible, we will only check the first rule of the ones mentioned hereinbefore when the algorithm is at the start of the roster. In a later stage, all rules will be checked.

### 4.2 Rostering for 6 weeks: a mathematical challenge

The algorithm as sketched in the previous section is rather simple and predictable. It guarantees nice individual schedules for every employee with a balanced workload and occurrences of free days and weekends.

However, preliminary experiments to obtain a good roster for 6 weeks have shown that it is difficult to find a general parameter setting which works well for all employees. The main problem that occurred was that the search space was too big for most employees when some default forwards rotation settings for all employees were used.
In order to give employees with a high appointment size (e.g. $\geq 32$ hours per week) a possibility to schedule their (hard) shift demands and soft shift demands (e.g. an employee who likes day shifts will likely have more preferable rosters when he gets a parameter set which allows for more day shifts and less late shifts), a flexible pattern was needed. This led to a tremendous number of roster possibilities for employees with a smaller appointment size or less restrictions. We have tried to overcome this problem by putting hard constraints on the employee's soft shift preferences (when an employee likes to have a certain day off or prefers a late shift on a certain day, the preference is assigned hard $)^{2}$, but this did not overcome the problem.

Creating a different generation setting to prevent the number of generated rosters, which depended on the size of the employees' appointment, also did not work out well. This occurred, because the problem of planning hard restrictions for some employees with a large appointment size now also turned up for the employees with a smaller appointment size. A parameter setting that worked for employees with a small appointment size with less constraints may lead to fewer rosters for some employees with (approximately) the same appointment size and more constraints (or vice versa for too much rosters for the least constrained employees).

We have tried to apply a waterfall method to bound the number of rosters:

1. Start with a very strict parameter setting.
2. When no rosters or only a small amount is found, relax some parameters to find more.

[^15]3. Iteratively repeat this until we have found a sufficient number of schedules.

This did not work very well, since the constraints for the several employees seemed to be rather different and it was difficult to find an order of relaxation steps that worked well.

This section will discuss the challenges and ideas of applying the basic algorithm, as sketched in the previous section, to obtain rosters for a period of 6 weeks. First, we will describe a few methods to find rosters by breaking up the roster period in two parts, thereafter we will describe a method to solve the full period at once.

### 4.2.1 Rostering 6 weeks in two steps

It was first tried to solve the problem by using blocks of 3 weeks, while generating with a relaxed parameter setting. In this way, it was likely that most employees (with a small or large appointment size) would obtain rosters and that the number of rosters for the least restricted employees was still a calculable number. We will discuss two methods.

### 4.2.1.1 Directly combining rosters for two times 3 weeks $(3-3)$

It was first tried to generate rosters for the first 3 weeks apart from the roster for the second 3 weeks and then to generate a roster for the 6 weeks as a whole by selecting a roster combination from a graph. The graph consisted of four layers:

1. A start node $s$.
2. All valid rosters for the first 3 weeks.
3. All valid rosters for the second 3 weeks.
4. A final node $t$.

All nodes in layer 2 were connected with $s$, all nodes in layer 3 with $t$. Furthermore, all valid combinations of rosters for the first and last 3 weeks were connected. A path $s \rightarrow t$ in the graph corresponded to a valid roster for 6 weeks.

Optionally, costs could be asserted to the edges in the graph: the costs of a roster $r_{1}$ in layer 1 could be added to the edge $\left(s, r_{1}\right)$, the cost of a roster $r_{2}$ in layer 2 could be added to the edge $\left(r_{2}, t\right)$. Furthermore, the cost on an edge $\left(r_{1}, r_{2}\right)$ could reflect the quality of the combination of rosters (how well does roster $r_{2}$ follow $r_{1}$ ). A nice benefit was that this method could be extended to find valid succeeding rosters, by iteratively introducing a new layer of 3 weeks, while fixing the roster for a lower layer number.

Example 4.2 (Finding succeeding rosters). First, a full schedule for 6 weeks is calculated, using layer 1 and 2. Then, a layer 3 is introduced for the next 3 weeks and the roster for the first 3 weeks (layer 1) is fixed. A roster for weeks $4, \ldots, 9$ (two half periods) is now calculated (Figure 4.2).

Thereafter, a layer 4 is introduced. The roster for the second 3 weeks (layer 2) is also fixed.
In this way, layer 1 and 2 form a valid schedule for the first six weeks, while a sequel for layer 3 and 4 is guaranteed.

Calculation of rosters for 3 weeks did not seem to be a problem, but creating a combination for both three week periods was. First, it was difficult to express when a connection between $r_{1}$ and $r_{2}$ was valid. It was possible to make a connection only if the resulting six weeks period perfectly fits within the shift sequence scheme (see page 46 for an example) and satisfies all predefined bounding rules, but of course also connections with small deviations from the pattern were possible.

However, a more important problem was the large amount of rosters in $r_{1}$ and $r_{2}$ that had to be compared to check whether the follow-up was valid and to store all the possible edges in the

| Period of |
| :--- |
| 3 weeks |

1 $\overbrace{3}^{\text {Roster period }} \overbrace{3}^{\text {Roster period }}$


Figure 4.2: Extension to calculate succeeding rosters. A roster for the first period of 3 weeks (layer 1) is already calculated and fixed. Currently, a roster for layer 2 and 3 is calculated. Thereafter, layer 2 will be fixed and a layer 4 (dotted line) will be introduced.
graph. Besides, this was a logical consequence when directly creating rosters for 6 weeks also raised generation problems.
4.2.1.2 Solving 3 week periods after each other, with-follow up guarantee $(3+1)-$ $(3+1)$

Not only a roster for 3 weeks seemed to be calculable, also the calculation of rosters for 4 weeks appeared to be more or less do-able. Instead of calculating the total employee schedule (with all staff demands) for a period of 6 weeks, the schedule is completely calculated for 4 weeks. Then, only the first 3 weeks of the calculated schedule are used for the final schedule, the last week is only a guarantee that a successive schedule exists.

Then, a new roster of 4 weeks is calculated, starting on the first day of the fourth week of the earlier calculated roster period. For this day (or the first 2 or 3 days), the shift assignment must be the same as in the calculated schedule for the first weeks. Since a shift assignment for the other days of this week was also found, it is likely that a kind of successive roster exists and hence the algorithm will find a roster for the second 4 weeks period, from which the first 3 weeks will be used (see Figure 4.3).

The main problem with this approach is its difficulty to efficiently express and combine the 6 week demands of the employees. For instance, the number of workdays during 6 weeks (except for holidays) is equal to the appointment size times the number of workdays per week with a deviation of 1 workday. How should the number of workdays during the first 3 weeks be defined? And how should we define the number of workdays during the first 4 weeks (thus including the guarantee week)? Half ( $\frac{2}{3}$ th respectively) of the 6 weeks' number with a deviation of 1 workday seems reasonable, but in practice this number may lead to suboptimality problems, especially when an employee requests some extra days off in a part of the schedule, while working a little bit more in the other part. Hence it might be better to increase the allowed workdays deviation during 3 or 4 weeks, but this leads to a lot of extra rosters which is what we like to prevent.

Furthermore, some choices can be made for the first 3 weeks, leading to a nice schedule, while giving troubles in the next 3 weeks. This may for instance occur with the night shifts. Since normally only 4 shifts per employee per 6 weeks are allowed, it might be the case that optimal


Figure 4.3: Calculating a roster for 6 weeks, using calculated rosters for 4 weeks. The final roster for the first six weeks consists of the first 3 weeks of the upper left roster and the first 3 weeks of the lower roster. The roster for the next period will start with the calculated week 7 of the upper right roster, while week 7 of the lower roster 'guarantees' the sequel.
choices for the first weeks lead to problems for the second period. For example: employees who did not have a night shift during the first three weeks are having a holiday during the second three weeks and hence are not available for the night shift. Hence it would have been better to assign the night shifts during the first weeks to them.

To conclude: it might be better to try to roster for 6 weeks at once. Luckily, a method which allows for this has been developed. It will be described in the next subsection.

### 4.2.2 Rostering 6 weeks at once by personalizing parameters

When splitting up the period into parts does not work well, the number of possible rosters for an employee should be increased/decreased (depending on the situation) using another method. The key observation in generating schedules is already implicitly made in the previous part of this chapter, but we will explicitly formulate it here: a method which works for a certain employee does not necessarily work for another. Hence, the major rule in generating employee schedules is thus individuality. Individuality does not necessarily imply that every employee should have his own roster generator or parameter set. It only means that small adjustments to a predefined parameter set can lead to a combination of both nice results and a limited number of rosters.

### 4.2.2.1 Introducing extra parameters and bounding rules

While testing with this observation, two more parameters have been introduced to allow for more types of generation systems. The old parameters do still exist, but for instance the late-day prevention, which was first regulated using a demand of at least 1 day off before the start of a day-block, can now be expressed by the new system. The new parameters are:

- A table of shift-following permissions. This table contains $3 \times 3=9$ cells with in each cell information whether workshift $w^{\prime}$ is allowed to directly follow workshift $w$ (YES), $w^{\prime}$ is allowed to follow $w$ when there is a set of free days between shift $w$ and $w^{\prime}$ (FREE_INBETWEEN) or $w^{\prime}$ is never allowed to follow $w(\mathrm{NO})$.
- A number of times the same shift block may be repeated. In the original system, it could never be the case that the employee has a day-block, then a free-block and then again a day-block, since there must be a late-block somewhere between the two day-blocks. The new system allows for a repetition of a workblock, after some days off, without being interleaved by a workblock of another shift type. To prevent an explosion of the search space, the number of repetitions is not unlimited. In most cases, no repetitions are needed at all. When it is needed, the allowed amount is mostly 1.

The additional bounding rules (each also holds at the start of the schedule) will now be defined as:

- Allocation of a workshift $w$ on day $d$ when there is a workshift $w^{\prime}\left(w^{\prime} \neq w\right)$ on day $d-1$ OR on a day $d^{\prime}\left(d^{\prime}<d\right)$ while there are only days between day $d^{\prime}$ and $d$, and workshifts $w$ and $w^{\prime}$ are not allowed to follow each other.
- Allocation of a workshift $w$ on day $d$ when there is a workshift $w^{\prime}\left(w^{\prime} \neq w\right)$ on day $d-1$ and workshifts $w$ and $w^{\prime}$ are only allowed to follow each other when there is a block of free days between days $d$ and $d^{\prime}$.
- Allocation of a workshift $w$ on day $d$, when the previous assigned workshift (on day $d^{\prime}$ ) is also of type $w$ and on there is a block of free days between days $d$ and $d^{\prime}$ and the number of repetitions is equal to the maximum.


### 4.2.2.2 Observations

Creating a setting that works for an employee might be a hassle, depending on the size of the appointment, the combination of preferences, fixed hard shift demands (e.g. an evening off every week) and incidental hard shift demands (birthdays, holidays, etc.). The total number of rosters per employee for a 6 weeks roster period should not exceed approximately 200.000 and is preferably more than 1000 (to have some choice). It may also occur that a specific parameter setting does not allow for night shifts. This might be hard and is sometimes impossible to fix, as can be seen in the next example.

Example 4.3 (Impossible night shifts). An employee works 3 days in a week, but does not want night shifts during weekends. On Friday, this worker has a fixed day shift for study reasons.

As explained in chapter 1, a night shift can only occur on the following days:

- Monday - Tuesday ${ }^{3}$
- Monday - Tuesday - Wednesday - Thursday
- Wednesday - Thursday
- Friday - Saturday - Sunday

Since every employee must have 2 days off after a night shift period, the only possibility to schedule the night shifts for the example employee is by creating the following week somewhere in the roster:

- A night shift on Monday and Tuesday
- A day off on Wednesday and Thursday
- A (study) day shift on Friday
- A day off on Saturday and Sunday

In this situation, the pattern should allow for a single day shift, which increases the number of possible rosters for the employee to millions. In case we do not schedule the weekend off, the employee has two more workdays in this week than his appointment size requires him to perform. This does not seem to be a pleasant situation.

Another alternative is to fix the full week somewhere in the roster (week $k$ ) and ignore the generation pattern for this week. Then, the generation system may generate for the first weeks $1, \ldots k-1$ and for the last weeks $k+1, \ldots n$, but this is currently not included in the model.
When an employee does not have a lot of changes in demands and preferences between different roster periods, then capturing the data and tweaking the parameter configuration have to be done only once and may then be reused in future roster periods. If it is needed to introduce a few incidental shift constraints during a roster period, among the recurrent ones, and the current

[^16]parameter set for the employee is not able to satisfy them, a small relaxation could be tried to make.

Unfortunately, the generation system remains a model. It is hard to cope with all the possible restrictions that can be invented. For some very restricted or special employees, it is even impossible to create a roster with this model (but this also holds for the two earlier mentioned systems). This may be obviated by using an alternative generation system for these employees (see section 4.4 and the discussion in chapter 7).

A still unresolved problem is the generation of rosters for employees with only 1 or 2 shifts per week. They can be assigned almost every possible shift combination. Hence it might be an idea to schedule these employees after calculating the full employee schedule for the other employees by performing a postprocessing strategy.
Finally, the system does not necessarily guarantee that a full schedule is created that satisfies all staff demands. Here too, a kind of postprocessing strategy might be good to apply to repair some errors in the final schedule. We will discuss some heuristics in chapter 6.
However, apart from these drawbacks, the model is very rich and allows for a lot of preference types. Preliminary experiments have shown that the solutions of this generation system in combination with the ILP as described in the previous chapter can get close to a solution satisfying all the hard constraints.

### 4.3 Soft requirements and costs of individual rosters

In case the depth-first generation pattern is used, a large number of different rosters is created. In a normal situation, approximately 200.000 rosters for a six weeks period is sufficient. Although all these rosters meet a lot of individual requirements, there is still a difference in quality.
Therefore we will introduce the concept of soft requirements. These soft requirements will be used to assign costs to the generated rosters in order to try to select a more preferred roster for the employee. Of course some cost functions could also be used for the special generators (e.g. the predefined weeks generator), but we will mostly use them for the depth-first generator.

Different kinds of cost functions can be designed. 'The sky is the limit', although it may be difficult to design the cost functions, such that the optimal mix of soft requirements is fulfilled. This section will describe the cost functions we use for an individual employee (with a DFS generation scheme) and we will introduce the concepts of scaling the roster costs and removing bad rosters in advance (as performed in Hoogeveen and Penninkx (2007)[17]).

### 4.3.1 Hard and soft shift requirements

A shift requirement is a requirement to (not) assign a shift on a particular day. Typically, this is a requirement for each week (e.g. no late shift on Wednesday or the Friday off), although it is possible to have them for some particular dates. We will distinguish two kinds of shift requirements:

Hard requirement: This requirement will always be granted. Only rosters satisfying the requirement will be created.

Soft requirement: This requirement is not guaranteed to be satisfied, but may influence the cost of the roster.

Most shift preferences of an employee are considered as 'hard'. This is done, since the UMC would like to take them into account seriously. By setting up a parameter setting for generating rosters which is able to create rosters which satisfy these requirements, the 'hard' variant can be used to reduce the number of rosters being generated.

Sometimes it is difficult to take a requirement into account (the generator is not able to generate rosters which satisfy the requirement every week or a combination of requirements is difficult) or
the requirement is not really important. In that case, a soft requirement could be used. This requirement is not guaranteed to be satisfied, but when the requirement is not satisfied in a roster, we could increase the cost of the roster. We could also provide a bonus (lower the cost) when the requirement is satisfied.

### 4.3.2 Other soft requirements

We will now introduce the other soft requirements for which a cost function is used. These functions are based on the ones in the list of preferences and demands (appendix C).

### 4.3.2.1 The preferred shift block length

Every employee may have a preference for the length of each shift block (specified as a minimum and maximum length). For instance, he may prefer to have large blocks of day shifts and small blocks of late shifts. When such a preference is infringed, a small cost per day shorter/longer can be added for the employee.
Note that this is something different than the hard generation constraint, defined in the earlier part of this chapter. The generation constraint is a hard bound on the minimum and maximum block length in the generated rosters, to guarantee enough rosters will be created. The concept discussed here is a preference. Hence it might be that a day shift block will be generated with a length of $2-5$ shifts, but that the employee prefers to have at most 3 shifts.

### 4.3.2.2 The preferred shift count

Every employee may also have a preference for allocation of more or less shifts of a particular type in his roster (specified as a minimum and maximum number). For instance, he may request for 'only a few late shifts', so preferably only 3 or 4 may be allocated during the period of 6 weeks.
The general idea is that having more unwanted shifts is worse than having less desired shifts. This holds especially for the sometimes very undesired night shift period.

### 4.3.2.3 Deviating from the number of workdays per week

One of the main phenomena occuring in most $24 / 7$ environments is that every week is a different one. This holds for the shifts being assigned, but also for the actual number of workdays in the week.

Every employee is appointed for a specific appointment size. As already mentioned, this size is equal to the number of workshifts to be performed each week. In reality, this is an average number. In the most ideal situation, the employee performs exactly this number of shifts every week, but this will almost never be the case in reality. Some weeks will typically be calmer than others.
To try to obtain a better balance between calm and heavy weeks, we will introduce a penalty on having more workdays in a week than average. The penalty will be given for every additional day above the average number.

### 4.3.3 Scaling the roster costs and removing bad rosters

All created rosters will have a cost value by now. Some will be relatively high and some will be very low. Since a lot of parameters can help increase the cost value, there is no general upper bound, which is difficult since the linear program defined in chapter 3 also works with additional costs to force a fulfillment of all occupancy demands and to try to obtain a nice spread of additional employees.

Therefore, the roster cost should obtain an upper bound, which means that the costs of all rosters created for an employee have to be scaled. Although any number can be used as an upper bound, we will use the value 1 . Hence, each roster $s$ for employee $j$ obtains a cost $c_{s}^{j} \in[0,1]$, reflecting how well it meets the employee's preferences.

The motivation here is that all generated rosters already satisfy a lot of hard constraints and most of them are already quite good therefore. The costs can be used in the (I)LP to try to select the best possible roster, but satisfaction of the occupancy demands is more important during that stage. By scaling the roster costs between 0 and 1 , the roster costs become very low compared to the cost parameters for shortness and surplus in the ILP.
Hence, the actual cost of a roster is hardly of interest while solving the (I)LP, since the (I)LP solution is largely determined by the aforementioned cost parameters for surplus and shortness. However, when there are some more-or-less comparable rosters available, which have the same effect on the surplus and shortness in shifts, the best ones will be selected.

We will prevent the selection of the worst rosters (the ones with the highest cost) by removing them before starting the ILP stage.
The process is as follows. After generating all possible rosters (according to the pattern) for employee $j$, the roster with the lowest and the one with the highest cost is determined first. Thereafter, we will scale the costs of all rosters.

- Let $c_{\max }^{j}$ be the cost of the roster for employee $j$ with the highest cost.
- Let $c_{\min }^{j}$ be the cost of the roster for employee $j$ with the lowest cost.
- Let $c_{s}^{j}$ be the cost of a roster $s$ for employee $j$.

If $c_{\min }^{j}=c_{\max }^{j}$, then $c_{s}^{j} \leftarrow 0$ for all rosters $s$ for employee $j$ (prevent dividing by 0 ). Otherwise we will change the cost of a roster $s$ :

$$
\begin{equation*}
c_{s}^{j} \leftarrow 1.0 \cdot \frac{c_{s}^{j}-c_{\min }^{j}}{c_{\max }^{j}-c_{\min }^{j}} \tag{4.1}
\end{equation*}
$$

We can now make a preselection by removing all generated schedules $s$ with $\operatorname{cost} c_{s}^{j}>0.5$.

### 4.4 Special generators for problematic employees

A few employees have restrictions which are unable to fit into the earlier defined Depth-first generation model. In this section, we will treat three alternative systems which might be used for them.

### 4.4.1 Predefined rosters

Predefined rosters can be useful for employees with a very limited number of rostering possibilities or for employees who like to perform self-scheduling. The full roster (or a few possible rosters) is directly inserted into the computer and will be used for this employee. It is possible to use the predefined roster as a base situation and to create a few mutations by changing a limited number of shifts in the roster (in case this is allowed for the employee). ${ }^{4}$.

### 4.4.2 Roster with predefined weeks

Some employees have a special regulation in which only a few roster possibilities during a week can be applied, but the number of possibilities is too much to enumerate them all by hand. Then,

[^17]the roster can be created using a so-called set of predefined weeks. We will recall the definition from section 1.7.2.1:

Definition 4.3 (Predefined Week). A description of a roster for a single week out of 6 weeks and how many times this week description must occur in the full period schedule. For example: "One week of the roster consists of a day shift on Monday, a day or late shift on Wednesday and a day shift during the weekend. In the final roster, there are two weeks for this employee which match with this description".

An example can also be found in section 1.7.2.1.
The algorithm will now create all possible different rosters with permutations of the above defined weeks, but it will not allow a day shift directly succeeding a late shift. As can be seen, it will be the case that many rosters with only one shift difference will exist. For instance: 5 weeks are the same in two rosters, but in the sixth week the day shift on Thursday is a late shift.

Clearly: the number of roster possibilities for a single week and the number of possible permutations must be very limited in order to apply this roster generator. Otherwise it will lead to millions of rosters for this single employee. Therefore, night shift periods will not be allocated by default.

### 4.4.3 No initial generation ('flexible generation')

When employees have a very small appointment size (a few staff members with a flexible contract) and are treated as fill-up when needed, it becomes impossible to enumerate all the possible rosters. There is no longer such a system as blocks of day shifts and late shifts. Hence, the number of roster possibilities for a single week is already quite large.

It may then be better to not roster these (flexible) employees initially and roster them when needed using a post-processing algorithm.

## Chapter 5

## ILP results and improvements

While the method was developed, the formation of the staff changed substantially. The most remarkable change was a decrease in the number of available employees (especially the SVKqualified employees). Besides, the UMC introduced some additional kinds of roster preferences (for instance: employees with predefined weeks, some employees appeared to have more demands than on the initial list, some employees appeared to have preferences which were different for even and odd weeks and some employees were having training days, some occupancy demands were slightly adapted). Furthermore, information about holidays and incidental shift requirements were not available for most roster periods.
The program has been rewritten a few times to cope with some of the newly introduced affairs. Therefore, it is hard to compare the initial roster calculations (on periods in the past) to those for the new periods, but of course these old calculations actually do say something.

During initial calculations, in which all employees were expected to be fully available (no holidays or special shift requests), the ILP as defined in section 3.4 and solved as defined in section 3.5 , was able to completely satisfy all occupancy demands. ${ }^{1}$ Moreover, most of the surplus was added to the day shifts. In the first runs, the first day shifts of the roster obtained a lot of surplus and the remaining part of the day shifts obtained a normal surplus, due to the way columns were introduced during a revised simplex step. This was logical actually, since there were more employees available than demanded (due to the expectation of no holidays). Therefore, the cost value $F^{\prime}$ was introduced (see page 38) to allow for a better spread of these supernumerary employees.

However, the solutions had a remarkable property. The difference between the optimal solution of the LP-relaxation and the solution of the ILP can be quite large. Furthermore, when the integrality gap is examined (the ratio between the final ILP solution and the best achievable fractional lower bound), we found that the gap appeared to be around $45 \%$ (after 15 minutes of solving the ILP). Solving a little bit longer (e.g. 30 minutes) did only drop the rate by a few tenths of a percentage. This could mean that:

- The result of the ILP is bad (compared to the LP-relaxation).
- The ILP solver had some major problems solving the ILP to optimality.
- The result of the LP-relaxation is bad.

Anyway, the most likely explanation is that some of the required rosters to efficiently solve the ILP were not available.

During the LP phase, it is allowed for an employee to have more than a single roster assigned, but each roster will be assigned only partially. The sum of these parts is equal to 1 . Hence a lot of occupancy demands for shifts are fulfilled using a group of employees who each perform the shift partially.

[^18]In the ILP phase, all parts of a shift will be assigned to a single employee who performs the full shift and will not have any other shift on the same day. This works very well for most shifts, but unfortunately, a (small) number of shifts could not be assigned to an employee out of this group. Perhaps there is an employee who still has some space to perform one of such shifts, but that is only possible if there is a schedule available for this employee which allows for this assignment. In case this schedule is not available, the shift will not be assigned.

Section 5.1 will introduce a mutation step which tries to resolve these situations by generating additional columns for the LP. In section 5.2, we will discuss some extensions to the ILP and the generation system in order to cope with some additional situations. Finally, a full result for a real period (including incidental shift requirements) will be presented in section 5.3. For this result, we used both the mutation step and some of the discussed extensions.

### 5.1 Adding roster mutations before solving the ILP

As already described in the introduction of this chapter, the reason for the remarkable worse ILP results and the high integrality gap is that the solution of the LP is fractional and employees may have several shifts on the same day partly instead of only 1 (e.g. two employees perform partly a day shift and partly a late shift). It could be that this problem is fixed by letting one employee perform the day shift and the other the late shift. Since a forwards rotating pattern is used for many employees, it may be that this particular roster is not available (created) and hence cannot be selected for solving the ILP.

It can therefore be a good idea to create a few additional rosters before optimizing the ILP with the help of the LP solution. These additional rosters may violate a few of the introduced restrictions (see below) and thus are new rosters that were not generated before. They are quite similar to existing rosters, but a few shifts have been changed. Therefore, we will call these new rosters 'mutations'.

Not all rosters will be mutated, only the most valuable ones. These are:

- All (partly) selected rosters. A roster $s$ is (partly) selected when $x_{s}>0$.
- The best rosters (according to the dual multipliers of the optimal solution of the LPrelaxation) which have been added to the LP-relaxation by a revised simplex step, but are not selected in the optimal solution of the LP-relaxation. Hence, each roster $s$ in this subset is represented by a column in the LP problem, but the corresponding value $x_{s}=0$.

The mutation operator is only applied on rosters which are generated using the forwards rotating pattern. A new roster is created using a combination of mutation operators (see the next subsection) and will thereafter be checked for validity.

A mutated roster is valid, when the depth-first roster generator (section 4.1) could have generated it in case the bounding rules were slightly softened. We will therefore not check whether the roster completely satisfies the defined generation pattern. Hence, all bounding rules as described in subsections 4.1.2 and 4.2 .2 will still be applied, but not the rules as described in subsection 4.1.3.

When the generator allows the new roster, it is added to a temporary roster pool and it will obtain a cost (see section 4.3). From this pool, the LP-solver selects the best rosters to use for the optimization problem. Then, revised simplex (adding columns to the LP) is restarted and some extra columns are introduced for solving the LP. Thereafter, the ILP is solved as normal.

### 5.1.1 Mutation operators

Two operators are used while performing mutations.

- Day-Late change Select a day shift in the roster and change it into a late shift or the other way around.
- Off-Work swap Select a day or late shift $\left(s_{1}\right)$ in the roster and a day off $\left(s_{2}\right)$. Then, shifts $s_{1}$ and $s_{2}$ are swapped. On the old position of the day off, a day shift and late shift will both be tried to insert.

These operators are executed on all possible shift combinations in the roster being mutated. For instance: the Off-Work swap operator is applied to all combinations of day/late shifts and days off in the roster.

Operators could be applied multiple times when we allow for more mutations on a roster. For instance: when we allow 2 mutations on a roster, we can perform two Off-Work swaps, but also an Off-Work swap and a Day-Late change.
To prevent the introduction of an already existing roster (as a result of mutation or as a result of reversing the mutation when multiple mutations are allowed), we assign a hash code to every roster we create. The hash code of each mutated roster will be compared to the code of all known rosters for the employee. In case the code is the same, both rosters will be entirely compared as a check, but in practice the rosters will definitely be the same and the mutated roster is not added.

### 5.1.2 Experiments and conclusion

We have performed some experiments on mutations. The setup of the experiments and the results can be found in appendix E .

It appeared that the integrality gap for the original experiment setting (assuming all employees are available, hence all shift demands could be satisfied) could be reduced to approximately $18 \%$. Furthermore, the objective value of the final ILP solution (the ILP solution when the time limit for solving the ILP was reached) was $34 \%$ lower ( 3301.51 instead of 5005.43).
Later, we also experienced that an even lower gap was possible in another roster period. The number of available employees was slightly less and for this period we had more information about holidays, teaching days and incidental shift requirements. Therefore, some (night) shifts could not be allocated. In this setting, we found that mutating using the best setup (but with 15 minutes for solving the ILP) was able to reduce the gap to approximately $4.5 \%$. We were satisfied with this result, until we tried to solve the ILP using no mutations at all. This time, we obtained an integrality gap of only $2.4 \%$. It can be expected that the gap will be lower when we allow a longer time period for solving the ILP.

Although the gap size is lower when no mutations are used, the solution itself is not better. It can be seen that the ILP value is $10 \%$ lower when mutations are used. Table 5.1 compares the results. The column 'LP value before' denotes the optimal value of the LP before adding any mutations, the column 'after' denotes the improved LP solution with mutated columns. The column 'Best bound' contains the best objective function value achievable at the time the ILP solving process was cut off due to a time limit. The column 'ILP value' denotes the value of the best integral solution at the time the ILP solving process was cut off. The gap is the difference between the latter two columns.

| Mutation | LP value |  | Best bound | ILP value | Gap |
| :---: | ---: | ---: | ---: | ---: | ---: |
|  | before | after |  |  |  |
| Yes | $72,996.31$ | $69,230.09$ | $77,363.59$ | $80,992.20$ | $4.48 \%$ |
| No | $72,996.31$ | - | $88,828.95$ | $90,975.91$ | $2.36 \%$ |

Table 5.1: Comparison of some mutation results.
We conclude that a correct mutation setting could help to reduce the objective value, hence to improve the result, considerably. Mutation also improves the integrality gap when all shift demands could be fulfilled, but this will not necessarily hold when the LP is not able to completely assign all occupancy demands to the available staff. The best results can be expected when the number of operations is low (only 1 mutation operation).

We should be careful not to add too many mutated columns to the ILP, since adding more columns will make the ILP optimization process considerably slower and the results (after a certain time period) worse. Moreover (stating the obvious) the results will be better when we allow a longer time for solving the ILP.

### 5.2 Some extensions of the method

The next subsections will introduce some extensions to the discussed method. Some of them have been implemented, others are suggestions for implementation.

### 5.2.1 Allocation of holidays

The allocation of holidays is quite easy, but we will discuss the concept for clearness. When an employee with appointment size $a$ (workdays per week) is on holidays, his holidays period typically consists of $v$ vacation days. Each vacation day replaces a normal workday. Hence, for a holiday period of 1 week, $v=a$ workdays will typically be replaced.

When an employee has a predefined roster, this is very easy: the provided roster consists of less workdays than normal and there is a large block of days off somewhere in the roster.
For the DFS-generation algorithm, hard shift requirements for a day off for each day of the holiday period will be inserted. Furthermore, we will allocate $v$ more days off than normal during the roster period of $k$ weeks, so only $a_{k}-v$ workdays will be inserted. ${ }^{2}$ In case it is needed, the number of work-weekends to allocate can be adjusted by the scheduling employee.

This concept works out of the box. There is no need to adjust a parameter set, since we will simply extend a default period of days off between two shift blocks. Because of the hard shift requirements, this can only occur in one position of the schedule. Since the fraction of workshifts to allocate is smaller, it is likely ${ }^{3}$ the number of possible rosters will decrease too.

It is also possible to allow a different sequel of the roster after the large chunk of days off. This can be done by checking less bounding rules in the first days after the holiday period, just like starting a new roster period (see definition 4.2 on page 47). However, we did not apply this step.

For employees with a predefined weeks roster, the concept is typically the same. Some of the predefined week blocks will change and hard shift requirements can be used to introduce the predefined 'holiday block' on the correct position of the roster.

### 5.2.2 Allocation of study days

Some employees have an incidental or weekly day on which they go to college for study reasons. The employee perceives such a shift as a day shift (hence he may for instance not have a late shift the day before). Therefore we will also treat it just like he has an ordinary day shift. However, we should take into account that the employee is not allowed to count for the occupancy demands.

When the employee has a study day, we must insert the day as a hard shift requirement, so the day shift will definitely be allocated on the particular day. We will then be certain the employee counts for all demand values (according to his qualifications) of the particular day shift $i$ in the ILP (the parameter $b_{i}$ and the like). Therefore, we will check the actual qualifications of the employee and then increase the demand for all $b_{i}$ values he influences. In this way, enough employees will be assigned to perform the normal work tasks.

We can use the same method when a senior nurse requests a quality day on a particular day.

[^19]
### 5.2.3 Allocation of quality days for senior nurses (SVK)

A possible method to allocate quality days for senior nurses is by using the occupancy demands. This especially works when there is a preference for allocation of such days on particular days in the period (e.g. preferably on Tuesdays), but this is not necessary. The idea is to select a number of suitable days for a quality day and then raise the SVK occupancy demand $\left(b_{i}^{\text {svk }}\right)$ for these days with 1 or 2 employees, such that more senior nurses will be allocated. The surplus could then be used to provide some of these employees a qualification day.

It should be noted that also the values $b_{i}^{\mathrm{mc} \vee \mathrm{vk}}, b_{i}^{\mathrm{mc}}$ and $b_{i}$ should be incremented with the same magnitude as $b_{i}^{\text {svk }}$. It is now possible that a superfluous MC nurse is allocated, since a senior nurse may not have the MC-qualification (only in exceptional cases). However, this is not a problem, because we just need some additional employees to satisfy the $b_{i}^{\mathrm{mc} \vee \mathrm{vk}}$ and $b_{i}$ demand anyway.

We could also adjust the ILP to insert a bonus for having some additional SVK employees on some days. A bonus parameter $e_{i}^{\text {svk }}$ is then introduced in equation (3.22) on page 38:

$$
\begin{equation*}
\sum_{s \in S} h_{j}^{\mathrm{svk}} d_{s i} a_{s j} x_{s}=b_{i}^{\mathrm{svk}}-z_{i}^{\mathrm{svk}}+e_{i}^{\mathrm{svk}}, \text { for each } i \in I\{\text { SVK demand }\} \tag{5.1}
\end{equation*}
$$

Since these employees may not be counted for the demand of 'lower' qualifications, we should also adjust the equations (3.21), (3.23), and (3.24). We will subtract the additional employees:

$$
\begin{align*}
& \sum_{s \in S} d_{s i} x_{s}=b_{i}+y_{i}+Y_{i}-e_{i}^{\mathrm{sk}}-z_{i}^{\mathrm{svk}}-z_{i}^{\mathrm{mc}}-z_{i}^{\mathrm{vk}}-z_{i} \text {, for each } i \in I \text { \{Total demand \}}  \tag{5.2}\\
& \sum_{s \in S} h_{j}^{\mathrm{mc}} d_{s i} a_{s j} x_{s} \geq b_{i}^{\mathrm{mc}}-e_{i}^{\mathrm{svk}}-z_{i}^{\mathrm{mc}}, \text { for each } i \in I\{\mathrm{MC} \text { demand }\}  \tag{5.3}\\
& \sum_{s \in S} h_{j}^{\mathrm{vk}} d_{s i} a_{s j} x_{s} \geq b_{i}^{\mathrm{mcvvk}}-e_{i}^{\mathrm{svk}}-z_{i}^{\mathrm{mc}}-z_{i}^{\mathrm{vk}}, \text { for each } i \in I \text { \{addable VK demand \}} \tag{5.4}
\end{align*}
$$

We can now insert an additional cost parameter $\varepsilon_{i}$ in the objective function (3.20) which has a negative value for the preferred days and 0 for the others. When $g_{i} \gg\left|\varepsilon_{i^{\prime}}\right|$ for all $\left(i, i^{\prime}\right) \in \mathcal{P}(I)$, we will be certain that allowing a quality day is only preferred in case all other employee demands are satisfied. The new objective function will be:

$$
\begin{equation*}
\text { Minimize: } \sum_{s \in S} c_{s} x_{s}+\sum_{i \in I}\left(\varepsilon_{i} e_{i}^{\mathrm{svk}}+f_{i} y_{i}+F_{i} Y_{i}+g_{i}\left(z_{i}^{\mathrm{svk}}+z_{i}^{\mathrm{mc}}+z_{i}^{\mathrm{vk}}+z_{i}\right)\right)\{\text { Total costs }\} \tag{5.5}
\end{equation*}
$$

Unfortunately the methods above will not work well when the number of available SVK-employees is quite low, since it will then already be hard to satisfy the normal SVK-demands.

### 5.2.4 Student - supervisor collaboration

An open problem is the collaboration of students with their supervisor (or in general: collaboration of two employees).

When it is needed to allocate the same schedule for a couple of weeks, we can do the following:

1. Create possible schedules for the supervisor (employee $e_{1}$ ) during the roster generation phase.
2. Create possible schedules for the student (employee $e_{2}$ ) during the roster generation phase, but do not provide workshifts during the collaboration period starting on day $d_{1}$ and ending on day $d_{c}$. Hence, this employee will obtain a period of days off between day $d_{1}$ and day $d_{c}$.
3. In case the employee $e_{2}$ also counts for the occupancy demands during the ILP phase, we will do the following: when employee $e_{1}$ obtains a workshift during the collaboration period, the ILP will normally count him as 1 employee assigned to the particular shift, but this time he will count for 2 employees (one time for his own qualifications and one time for all qualifications of employee $e_{2}$ ).
4. After solving the ILP, all workshifts for employee $e_{1}$ during the collaboration period will also be assigned to employee $e_{2}$.

To overcome problems with Late $\rightarrow$ Day transitions and a too small buffer period after a possibly allocated night shift block (only when employees $e_{1}$ and $e_{2}$ are not students), two additional days off on day $d_{c+1}$ and $d_{c+2}$ should be allocated while generating.
Based on the experiences with programming the main roster generation system, this will already be very difficult to program well, although it might be possible. However, when both employees have a couple of shift requirements (the student has a study day, the accompanist has some requests for days off), it will (almost) be impossible to use the current roster generation system, since it will be quite undoable to find a nice generation scheme.
When it is only needed to have a predetermined fraction of overlapping shifts, it might be that this is automatically satisfied after solving the ILP. When the collaborating couple consists of two employees with a high appointment size, we will be quite certain especially many day shifts will be allocated in their roster. There will definitely be some days on which they perform the same shift. In case more shifts are needed, a change can be made afterwards.

Furthermore, some collaboration days can be appointed in advance. For these days, both employees will have a hard shift requirement for a work shift. Hence they will be allocated to the same shifts on these days.

### 5.2.5 Successive schedules

This final extension describes a suggestion to introduce successive schedules. The concept of successive schedules consists of two parts:

1. Guaranteeing a schedule for period $p+1$ exists, when period $p$ is generated.
2. Guaranteeing a schedule being generated for period $p$ sequels to the schedule of period $p-1$.

Although the concepts look rather the same, this is a misconception. A schedule generated for period $p$ is likely to change. The scheduling employee will perform some manual changes immediately after generation, in order to let the schedule satisfy some demands which could not be introduced directly. Besides, the schedule is likely to change during period $p$, since there will always be some last minute changes needed (e.g. employees may get ill and the nurses in the UMC perform a lot of shift exchanges).

Therefore, when rule 1 is satisfied while solving the ILP, it will not ensure that a strict (according to the generation pattern) follow-up schedule remains to exist in the end. Furthermore, many aspects of the schedule for the next period are not known in advance. Hence the connection to the schedule of period $p-1$ should still be checked, while generating the one for period $p$ (rule 2).

### 5.2.5.1 Guaranteeing a successive schedule (Rule 1)

While developing the methods, we have already found that guaranteeing a full extensive roster is not easy. The (probably) best possible way to do this, is as performed in Hoogeveen and Penninkx (2007) [17]:

1. Calculate a schedule for two periods at once using a generation system for two periods (12 weeks).
2. Fix the schedule of the first period ( 6 weeks).
3. When the final schedule for the second period must be calculated: calculate a schedule for the periods 2 and 3 , such that it 'fits' the schedule of period 1 (see section 5.2.5.2).

However, we currently could not calculate a schedule for two periods, since one full period of 6 weeks is already challenging.

Suppose we were able to make a calculation like this. It is likely that both the schedules for the first and second period will be good after the first calculation, since both schedules are calculated at once. However, it could be the case that the schedule for the first period will be better when it is not directly calculated with the schedule for the second period (but this will then have a negative effect on the latter schedule). Furthermore, the succeeding schedules (for periods 3, 4, etc.) will likely also be good (although not necessarily optimal), for the same reason.

But as already said, it will be impossible to calculate this, since the number of possible rosters for a single roster period is much larger compared to the earlier mentioned paper.

A second possibility is to guarantee that every individual roster being generated has a successor. For a roster being calculated using DFS this can be achieved as follows:

1. Calculate a roster for $k$ weeks as normal.
2. Extend the roster for $d$ days (for instance: a week). When this is successful, it is likely that a connectable roster for the next period will also exist.
An important open question here is: how many workdays should be allocated during the $d$ extra days? (A fraction $\frac{d}{7}$ of) the normal appointment size of the employee? Do we allow a single workday more or less during the week? Do we allow the next weekend to be off? Also when these questions are answered, there is no real guarantee a follow-up schedule exists.
Since many aspects of the roster will change after the ILP has been performed and information about a next scheduling period is hardly known, this approach will not really provide more information than the situation in which a follow-up is not guaranteed at all. It might be better to move the investigations to the second rule.

### 5.2.5.2 Guaranteeing a schedule is successive (Rule 2)

At the time a new roster is generated, most information for the new period is already known. Furthermore, the previous period is almost halfway. Therefore, most of the shift exchanges requested by the nurses have been allocated. Hence, it is largely known how the period will end (except some possible changes due to illness).

Basically, we have two choices for the start of period 2:

1. Try to continue with the pattern of the previous period.
2. Restart the pattern in the new period, but perform some additional checks.

The first choice is only an option when a schedule of an employee has not been changed during the first period and the pattern remains the same during the second period. This is not a likely situation. Moreover, nobody really cares whether the pattern is exactly extended into the second period. We can just start generating schedules like we did for the first period, but we should bear in mind that the transition from the first to the second period must be valid.

Definition 5.1 (Valid schedule transition). A schedule transition is valid when:

1. We do not introduce a transition from a late shift to a day shift.
2. We do not violate a buffer period after a block of night shifts.

Both constraints can be inserted manually for all employees by the scheduling employee, but it is also possible to read the last shifts of the previous roster period and let the program introduce these constraints automatically.
A valid schedule transition does not necessarily mean it is desired too. Therefore, we should also make sure:

1. We do not introduce a long period of workdays.
2. When the employee performed a night shift block during the last week of the previous period, we do not introduce a new one during the first week of the new schedule.
3. We do not introduce too many consecutive workweekends.

The second rule can be enforced manually, by forbidding a night shift during the first week of the new roster.

For the first rule, it is possible to insert a hard shift requirement to allocate a day off on the first Tuesday or Wednesday. We will then be sure that not too many consecutive workdays will be allocated. The third rule can also be satisfied by using a hard shift requirement (allocate the first weekend of the new roster off).

Also this time, it is possible to perform these checks automatically. However, although it looks easy at first sight, it is not. The effects on the roster generation system are very unclear. It may be that the generation scheme has to be changed in order to generate (enough) rosters which satisfy both the 'transition constraints' and all (incidental) shift requirements. This is an undesired situation, since you do not want to adjust the scheme in every roster being generated. Furthermore, it is preferred to have a generation systems which is guaranteed to work 'out of the box', but this is very hard to develop. A lot of exceptions and options must be programmed in order to let the program cope with exceptional situations.
On the other hand, it remains to be seen whether the additional 'desired' rules are really needed in practice. When the roster pattern of an employee does not change significantly, it is likely that most problems will only occur rarely. Hence, only guaranteeing the rules in definition 5.1 might be enough in practice. In case an undesired situation is introduced, a manual fix can be applied, probably using an automatic repair method (chapter 6).

### 5.3 Results for February - March 2013

We have made a big calculation for a 6 week period in February - March 2013. For this period, an up to date list of employee preferences and training days was available. We also obtained a list of employees with a predefined (weeks) roster and all incidental shift requirements and holidays of the employees. The data is shown in appendix C.

Although flexible employees are meant to fill up the final occupancy problems, some of them have been scheduled directly using the ILP. This holds for the employees with a predefined roster (selfscheduling) and the ones with a rather large appointment size (they are scheduled using the DFS generation system). The remaining flexible employees did only obtain a roster consisting of days off and may be scheduled later (see chapter 6).

An extended description of all parameter values and some example employee patterns can be found in appendix F.

### 5.3.1 Constraints that have (not) been used

We have inserted almost all shift requirements as a hard constraint for generating the rosters (except for a constraint which contained the word 'liefst' (preferably), which was added as a soft constraint). We experienced some problems with inserting a request for the Friday off for SVK employee 4, hence this should be repaired afterwards (the demand was inserted as a soft requirement). This kind of problems is related to the following combination of (general) requirements, which make it difficult to find a good set of roster parameters:

- This employee should work during 3 weekends and have exactly 3 weekends off in the roster period.
- The appointment size of the employee consists of 4 days a week.
- A block of days off should be at least 2 days for employees with a large appointment size.
- The employee should not perform too many consecutive work shifts.
- The number of generated rosters should not be small and not be too big.

Typically, when the employee works during a weekend, he should have a block of 2 days off somewhere in the previous and next week. Then, having all Fridays off in the roster period is incompatible with this demand.

Also requests for working odd weekends and having even weekends off (or the other way around) are sometimes difficult to allow in combination with all other demands.

A request for the length of a shift block (e.g. 'max 3 consecutive night shifts', see section 4.3.2.1) was inserted as a soft constraint, by preferring for instance blocks of at least 0 and at most 3 night shifts (so blocks of 4 shifts could be generated, but may be dropped by the preselection operation).
A request for the number of shifts in a roster (e.g. 'the number of night shifts as small as possible') was also inserted as a soft constraint (see section 4.3.2.2). Such requests could also be formulated differently. For example: a request for 'at most 2 late shifts per week', was translated to a soft preference for allocating at most 12 late shifts per roster of 6 weeks.

The rosters were generated and solved using most of the demands described in chapter 1 . The following hard or soft constraints were not (fully) taken into account:

- Assigning quality days to senior nurses (except when the employee has requested a particular day in advance).
- Collaboration with a supervisor.
- Having no single days off. For some employees we allowed a single day off in order to grant a shift requirement (e.g. no late shifts on Tuesday).
- Having 3 days off after a night shift block whenever possible.
- Having a late shift on Friday when the employee performs a late shift during the weekend (difficult to use with the shift block lengths).
- Requests for odd or even workweekends.
- Having a nice spread of the night shift blocks. Although it is not guaranteed, it turned out to be no problem in practice.

Furthermore, we did not check whether an employee's roster connects well to the previous roster period.

### 5.3.2 The results

The results for this roster period can be found in appendix F. The first thing to notice is almost all occupancy demands have been satisfied!

The most important problems arise during the night shifts in the weekend. Typically, the algorithm should assign 2 or 3 more employees to these shifts during the full roster period, but is unable to do this. There are three important reasons for this behavior:

- The pool of employees to assign night shifts to is a little bit too small for the requirement that every employee may perform at most 4 night shifts. Since a weekend night shift period consists of 3 days, an employee having such a period may not have any other night shift period and the fourth allowed night shift can therefore not be allocated. For the optimization algorithm, it is therefore a better choice to satisfy all weekly night shift demands, since every employee could then obtain two night shift periods of 2 days or 1 night shift period of 4 days. The algorithm then consumes the full 'space' of 4 allowed night shifts per employee.
- Many employees have a preference for day or late shifts during the weekends they work. Furthermore, every employee may only work a limited number of weekends.
- Some employees with a predefined roster or predefined weeks roster did not obtain any night shift. However, most of them may obtain a night shift period.

On the other hand, some of the employees with a predefined roster assign themselves more than 4 night shifts during a roster period. This reduces the extent of the problem.

Furthermore, there are some problems with the demands of the SVK-qualified employees during the day shifts. This is more a result of the small number of available Senior nurses in this period ( 5 instead of 7 in other periods we have examined). Most of the problems arise on the days with a higher demand than normal: 3 instead of 2 employees. This higher demand is the result of an employee who requested some quality days in advance.

When the rosters of the employees (only the ones which were generated and mutated using the depth-first pattern) are examined, we see that work periods and free periods are interspersed quite well. Many employees with smaller appointment sizes have work blocks in which they work the number of shifts as demanded by their appointment size. Thereafter, they will then have a large block of days off. For most employees a block of days off always decently consists of at least 2 consecutive days.

The weight of the work weeks is balanced well in general, although sometimes a little unbalance may occur (see VK employee 11 in the second solution for an example, this employee has a large chunk of days off in his schedule). Furthermore, the rosters follow a nice forwards-rotating pattern (with sometimes a period of days off in between) and work weekends are spread well.

Most important fact is that a lot of hard and soft individual requirements have been allocated. Firstly, most of the shift requirements (preferences for days off or a certain shift on a certain day) have been allocated since they were introduced as a hard requirement during the generation step of our solution approach. Furthermore no employee will experience a combination of a lot of other undesirabilities, since the algorithm removes rosters with a high cost in advance.

### 5.3.3 Conclusion

To conclude: we are quite satisfied with these results! The algorithm performs very well on the test case.

However, we should note that setting up the generation patterns could be a hassle. There is also an upper bound on the number of constraints that can be taken into account when the rosters are scheduled this way. As already mentioned, some of the constraints as defined in chapter 1 have not been taken into account.

Furthermore, the UMC asked in a later stadium to not or mostly allocate late shifts to some of the employees. This is however difficult to realize using the pattern as discussed, since the alternation of day and late shift blocks and the natural days off between them suppresses the number of generated rosters. It is impossible to allow a day or late block to consist of only a single shift, since the search space will then explode (an exception arises in a rare case, when a flexible employee has a very limited number of possible shift allocations on most days).
Hence, although the results are quite good, manual tweaking will always be needed afterwards. But, you could better perform a few manual tweaks afterwards than creating the full schedule by hand.

## Chapter 6

## Heuristics to repair imperfections

Once the full schedule has been calculated using the ILP approach, it may still have a few imperfections. As can be seen in the results (chapter 5), the quality of the employee schedules is rather good. The employees will not have too long periods of consecutive workdays and a lot of their shift preferences (e.g. a late shift on a certain day) will be assigned. Furthermore, most surplus of employees is added to the day shifts.

As a consequence of using a roster generation scheme for each individual employee (which reduces the number of possible rosters) and all the preferences, it might be the case that some occupancy demands can not be fulfilled completely by the ILP. Furthermore, some employees with a flexible contract may still not have been assigned. Moreover, not all demands can be (easily) taken into account during the roster generation phase and the ILP phase. For instance: the concept of employee collaboration is hard to introduce.

The scheduling employee may try to resolve all these problems himself, but a computer may also perform a part of this process or provide some suggestions.
This chapter will introduce a system of heuristics which tries to repair imperfections in the roster resulting from the ILP phase. Furthermore, this new system is able to process roster changes on request (i.e. providing a particular shift on a particular day for a certain employee). The key observation behind the repair heuristics is that we allow the computer to 'slightly' change the schedules of employees, such that they may obtain a 'mutated' schedule. These mutated schedules were not valid during the roster generation phase (and therefore not created during the generation), but we will allow them now, provided that the changes are carefully made.

We will try to repair the following types of imperfections:

- Some hard shift requirements (days off and preferred shifts on a particular day) for a single employee, which have not been allocated. The most likely reasons for these problems are:
- An employee who must have a particular shift on a particular day, but it was impossible to include it in the generation parameter setting.
- The requirement was introduced after the roster calculation process was finished.
- Employee shortage in some shifts.
- A change in the number of workdays an employee should obtain during the rostering period.
- (Partly) allocation of shifts to employees with a flexible contract.
- Some suggestions for other desirabilities, such as introducing quality days for senior nurses and student - supervisor collaboration.

Some of these problems are more difficult to repair than others. For instance, assigning an employee to a night shift period afterwards (in order to resolve shortage) may involve a change of shifts on 5 days, since a weekend period always consists of three consecutive night shifts and at least two
consecutive days off. This may have a big effect on the overall quality of the roster for the considered employee.

On the other hand, a shortage during a late shift on Wednesday might be quite easy to resolve, since we could assign a superfluous employee from the day shift (although we may not introduce a Late $\rightarrow$ Day transition during this action).
Section 6.1 will further introduce the concept of repairing. We will provide some definitions and new notation and discuss the building blocks and restrictions of the system.

As already described, some problems can be repaired more easily than others. Section 6.2 will therefore introduce and motivate a repair order.
Section 6.3 already contains some heuristics. However, the ones defined here are some important auxiliary functions which can be used in the subsequent sections. These auxiliary functions are mainly used to resolve some problems which can be introduced by other heuristics.

The sections 6.4, 6.5 and 6.6 contain the most important heuristics. These are the ones for repairing currently unsatisfied hard shift requirements (preferences for a day off or a particular shift on a particular day) and resolving shortages of employees.

Also section 6.7 contains some heuristics. These are used to introduce other (less important) desired properties in the rosters. We will call them 'extensions'.
This chapter will be concluded with some experimental results in section 6.8.

### 6.1 Introduction and definitions

The main ILP approach is designed such that all employee overflow will be put in the day shifts (especially during weekdays). As can be seen in the results, this actually happens as desired. Except for a rare situation, all overflow is put in the day shifts of both weekdays and weekends.

The key observation behind the repair algorithm is that we could use these day shifts as a buffer. When there is a shortness in a particular late or night shift, we may retract an employee from a day shift on that particular day (or we can change his day off in a workday and change another day shift in a day off).
When all shortness issues with night shifts and late shifts (whenever possible) have been resolved, there is a high probability that there will be no shortness issues with day shifts (except for the SVK demand, since the number of available senior nurses is currently small). When there are some remaining issues, we can try to change a late shift with surplus in a day shift or (only during weekdays) swap the day off for an employee.
Before the algorithm is executed, the desired number of workdays for each employee should be clearly inserted into the system. By default, the algorithm will keep the (in the ILP phase) allocated number of workdays ${ }^{1}$. When the scheduling employee wants to change the allocated number for a particular employee, he may change the number of allocatable workdays and the algorithm will try to remove or insert extra days.
Furthermore, all demands of the flexible employees should be inserted well into the system. Whenever possible, the repair algorithm will first try to repair an occupancy shortness in a shift by assigning a flexible employee. It turned out that flexible employees will never perform night shifts.
We would like to mention that a lot of repair operations seem to look rather obvious, but this may partly be a false impression. Designing and implementing them such that they provide predictable results can be a hassle. A deep understanding of rosters which have quality was needed for developing them, since many aspects (e.g. hard requirements, shift sequences, and lengths of work periods) should be taken into account. We have tried to break all complex actions

[^20]up into smaller operations, but regularly it appeared that identifying these smaller activities was not obvious.

This section will provide an introduction to repairing rosters. The information is important in order to understand the contents of the next part of this chapter. We will firstly introduce a new system of shift requirements in subsection 6.1.1. Thereafter, subsection 6.1 .2 will discuss some important restrictions which hold for repairing rosters in general. Subsection 6.1.3 will introduce some new notations. Finally, subsection 6.1.4 discusses the major building blocks of most repair heuristics defined later in this chapter.

### 6.1.1 A new definition of a shift requirement

We will change the old definition of a shift requirement as defined in section 4.3.1. The old definition and its associated problems is discussed first. Thereafter, we will present the new definition.

### 6.1.1.1 Old definition of a shift requirement

In the previous chapters, two different kinds of shift requirements (a 'preference' or constraint for allocation of a shift on a particular day) have been considered:
Hard requirement: Is used while generating (and mutating) rosters and will always be granted.
Soft requirement: Is used while generating (and mutating) rosters, but will only be used to adjust the cost of a roster. Some rosters will then be less attractive to introduce as a new column in the ILP. Furthermore, it is used to remove a selection of substantially undesired rosters (see section 4.3.3).
Unfortunately, this number of requirements is insufficient for the repair phase. The hard requirements were used to reduce the number of generated rosters, while guaranteeing that most preferences of employees will be allocated. This means that a lot of 'preferences' for an employee were actually included as hard requirements and that all generated rosters for an employee satisfy all these hard requirements.
However, a preference for a particular shift on a particular day is less important than satisfaction of the occupancy demands. It is likely that a lot of shifts could obtain the correct number of assigned employees, when some of the employees will not obtain a few of the initially assigned shift preferences. This could be resolved by using mainly soft requirements during generation instead of hard requirements, but we have experienced that the resulting employee rosters will then have less quality. Namely, the following would have happened:

1. The number of generated rosters becomes huge (most times incalculable or impossible to store in memory).
2. To overcome this, a stricter parameter setting must be used.
3. A stricter parameter setting may result in rosters in which less employee preferences could be granted.
4. Hence, the general quality of the employee schedules becomes (much) worse.

Hence, we prefer to include most preferences as hard requirements during the generation phase, but we should be able to soften most of them to help improve the satisfaction of the occupancy demands during the repair phase.

It must be observed that the various hard requirements are not all equally 'hard'. Some employees have more preferences than others and not all preferences are equally important.

We should therefore introduce a requirement system which allows us to indicate which hard requirements may be softened during roster repairing and which ones not. Furthermore, it is not always possible to introduce all obliged shift requirements for an employee during the generation
phase, so we should also add the possibility to have a requirement which is only taken into account during the repair phase.

### 6.1.1.2 New definition of a shift requirement

Four kinds of shift requirements will be distinguished from now on. Any possible requirement falls into exactly one category. The kinds of requirements are as follows:
Soft requirement: Is used as described above while generating (and mutating) rosters. It will not be used during the repair phase.

Hard generation requirement: Is used as a hard requirement while generating (and mutating) rosters, but may be overruled during the repair phase.
Hard permanent requirement: Is used as a hard requirement while generating (and mutating) rosters and will normally ${ }^{2}$ not be overruled during the repair phase.

Hard repair requirement: These requirements will not be taken into account while generating (and mutating) rosters, but will be realized during the repair phase. Not all types of requirements can be granted by the repair algorithm. The restrictions will be defined below.
The hard generation requirement is handy to use for fixed shift restrictions (e.g. a preferred day, evening, or weekend off). When the employee has a small number of restrictions, the requirement may be set as permanent. In this way, it is certain that the employee will have and keep his preferences.
However, when an employee has a lot of preferences, it might be unfair to allocate them all, when problems with the occupancy in a shift need to be resolved and lead to bad rosters for other employees. Therefore, it is better to set the preferences of these employees as hard during the generation phase only, so they may be overruled during the repair phase. In practice, the number of required repair actions is limited. Hence, the number of changes during the repair phase is small and most of the preferences will still be kept.
The hard permanent requirement can in general be used for incidental occurrences. For instance, when the employee has a training day or is having a holiday it is not allowed to change his shift allocation for a specific day. It is also possible to prevent the assignment of a night shift on particular days.
The hard repair requirement is hardly used. The possibilities to use it are limited (see below), but it can be helpful in case a requirement can not be added in the generation phase. This is most likely due to an overflow of requirements for an employee or a strange shift pattern demand (e.g. a full week of consecutive late shifts instead of only 1 or 2 ). Furthermore, it can be used to insert a requirement after the roster has been calculated.
For every possible shift requirement of an employee, the scheduling employee can decide into which category it will fall.

### 6.1.2 Restrictions for the repair system and the new shift requirements

As discussed in the introduction of this chapter, the repair system is used to satisfy some really hard requirements (occupancy demands and some shift assignments for employees) which could not be (fully) taken into account during the generation and ILP phase. In case a requirement can already be included during the latter phases, it is always better to do so. Repairing the roster afterwards to introduce a desired situation is worse than taking it into account while calculating the roster from the start.

Besides satisfying some really hard requirements, another purpose of the repair system is to try to perform the required roster changes without making the overall roster quality much worse. We must therefore compose a few restrictions for the repair system.

[^21]Some restrictions are easier to be kept or introduced by the repair system than others. The first observation is of course that conflicting combinations of hard requirements (as introduced in the previous subsection) may lead to unpredictable effects. Hence, the scheduling employee should only insert realistic shift requirements.
Furthermore, the basic limitations of the repair system are as follows (the motivations will be discussed below):

- The repair system may assign night shifts to employees to resolve occupancy problems. However, it is not possible to let an employee request a night shift period during the repair phase by introducing a hard repair shift requirement (section 6.1.1.2) for a night shift.
- We will not change a workweekend into a weekend off or the other way around. A requirement for a weekend (not) off must therefore already be inserted during the generation phase. Although we can change an assigned day shift during a weekend into a late shift (or the other way around), it is preferred - but not obliged - to introduce such hard shift requirements during the generation phase.
- When a hard repair requirement (section 6.1.1.2) is used, it may only be a requirement for a shift that must be allocated. Hence, it might not be a requirement to not allocate a specific shift on a particular day.

Hence, among repairing shortnesses in shifts, the repair system can be used to introduce a hard repair requirement (section 6.1.1.2) for:

- A particular day off.
- Allocation of a day shift on a particular day.
- Allocation of a late shift on a particular day.

Other types of hard repair requirements will not be taken into account.
While the system performs the heuristics to repair shortnesses and introduce the hard repair requirements, the hard permanent shift requirements (section 6.1.1.2) will normally be kept. There is an exception for a few situations, in which a hard permanent requirement for a night shift or for not having a particular shift on a particular day may eventually be overruled by the repair system.
The remaining part of this subsection will extensively describe the constraints and motivations.

### 6.1.2.1 Night shifts

A hard repair requirement for allocating a night period will be ignored.
Although it is hypothetically possible to request a night shift on a particular day, it will rarely occur in reality. The only likely reason is that an employee has drafted his own schedule and has allocated some night shift periods. However, this employee may be indifferent about changing a night period.

Introducing a requested night period for an employee during the repair phase could be quite difficult, since an already allocated period might be changed or retracted, due to the maximal number of allowed night periods for the employee. Furthermore, retraction of a night shift period is something we do not want to do, since many days in the roster (the night shift period itself and possibly the two days off afterwards) should be changed. The roster quality may therefore drop considerably.
Besides, when an already allocated night shift period is retracted, we have introduced an additional shortness during a night shift period, among the already existing shortnesses during night shift periods as a result of not completely satisfying the night shift occupancy demands during the ILP phase. We should assign another employee to this newly introduced night shift period with shortness. Since assigning a night shift period to an employee may involve a change of at least 4 consecutive shifts in a roster, it could have a huge (bad) influence on the roster quality.

Therefore, we would like to prevent night shift allocations during the repair phase (and hence introducing night shift shortnesses by retracting night shift periods) as much as possible.

### 6.1.2.2 Weekends off and workweekends

It is not possible to change a workweekend into a weekend off or vice versa. The reason for this is that a weekend can be seen as a natural roster break. Allocating an initial weekend off as a workweekend (or vice versa) entails a corresponding change of another weekend too.

It is likely that an employee with an (almost) full appointment size will obtain a very long work period when his natural 'break' weekend is changed into a work weekend or a very long period off in the other case. Solving this problem is quite hard. Therefore, a request for a weekend should be inserted in the generation phase.
Nevertheless, an assigned day shift during a weekend may of course be changed into a late shift or the other way around.

### 6.1.2.3 Requirements for not allocating a particular shift

It is possible to have a hard permanent requirement for not allocating a particular shift on a particular day. The repair algorithm will strictly retain this requirement for a night shift (i.e. do not introduce a night shift for an employee on a particular day), but may infringe it for other shift types in case a scheduled night shift period has to be retracted, due to a hard repair requirement.

Example 6.1 (Infringing a hard permanent requirement). An employee has a permanent requirement for not having a day shift or day off on a particular Monday. Therefore, he has obtained a roster in which a night shift is assigned on both Monday and Tuesday. However, according to a hard repair requirement, the Tuesday should become a day shift.

Since night shifts have special restrictions (see section 1.4.1), also the shift on Monday must therefore be changed. A day shift and day off are forbidden, so only a late shift may be assigned on this day. However, this introduces a Late $\rightarrow$ Day transition on Monday and Tuesday, which is illegal.

In this case, there is no valid solution applicable. Hence, the algorithm must then deliver a solution with a small irregularity and could introduce a day off on Monday (which was declared to be forbidden).
It is likely that such situations will rarely occur. In case it occurs, we will allow the algorithm to ignore the hard permanent shift requirements of the type 'do not provide a particular shift on a particular day'. This is only needed when a night shift period is retracted. In case the algorithm introduces an undesired situation, the scheduling employee can fix it automatically by introducing a hard repair requirement and execute the repair algorithm again or perform a manual change afterwards. In the example, he could insert a hard repair requirement to allocate a day shift on Monday.

Besides, such a situation may be prevented during the generation phase. When the hard repair requirement (e.g. a day shift on Tuesday) is already known during generation, but could not be included as a hard generation requirement, it is possible to introduce a hard generation requirement to not allocate a night shift on Tuesday. Then, the night shift period will not be given and this problem will not occur.

Again, due to the likely small number of repair requirements and the small number of night shifts in a schedule, the chance this kind of problems will actually occur is rather small.

### 6.1.3 Additional notation in this chapter

In the next sections of this chapter, we will use some (new) variables. We will define or repeat them in this subsection.

### 6.1.3.1 Definition of $a_{j}$ for employee $j$

With $a_{j}$ we denote the appointment size of employee $j$ (could be fractional, e.g. 3.5 days per week).

### 6.1.3.2 Definition of $\eta_{j}$ for employee $j$

Some employees create their own full schedule for the rostering period, which is then (approximately) adopted by the scheduling employee. It is possible (and it actually occurs) that they allocate more night shifts than is actually allowed by the general rules (for most employees: at most 4 night shifts). We do not want to change this in the repair phase, since the scheduling employee will have benefits from these nice employees.

Therefore, we will introduce a parameter $\eta_{j}$ :

- Let $\sigma_{j}$ be the number of allocated night shifts for employee $j$ during a rostering period.
- Let $\varsigma_{j}$ be the highest number of allowed night shifts for employee $j$ during a rostering period (according to the general rules).
- Let $\eta_{j}$ then be defined as $\max \left(\sigma_{j}, \varsigma_{j}\right)$ : the highest number of night shifts employee $j$ may obtain during the rostering period.


### 6.1.3.3 Definition of $w_{j}$ for employee $j$

The parameter $w_{j}=\left\lceil a_{j}+1\right\rceil$ will be used to define the desired number of consecutive workdays for employee $j$. In other words: we will define the desired number of consecutive workdays as one more than the average number of workdays during a week.

A necessary remark is that the schedules may deviate from this rule, since some employees tend to create their own schedule and hence may introduce longer work periods.

### 6.1.4 Building blocks of a solution method

In many of the resolution methods defined in this chapter, three steps can be distinguished:
Preselection: In this step, all employees who can be used for the action are selected and stored in a list. When the size of the list is 0 after the preselection step, the action cannot be performed.
Sorting: For some employees, it is easier (or more desired) to use them for the action instead of others. In this step, the preselection list is sorted, such that the 'best' employee to use for the action is at the top of the list. This employee is then chosen to use in the action.

The sorting grounds are mainly based on the (SVK) qualifications of the employees, the (expected) number of changes needed in an employee's schedule and the estimated quality of the employee's schedule when the action has been performed.

Each employee $x$ is compared with each other employee $y$ from the first (smallest number) till the last rule. When employee $x$ meets a particular rule and employee $y$ does not, employee $x$ is preferred over $y$ and the rules specified thereafter are not checked for the combination of $x$ and $y$. Otherwise, the algorithm will examine the next rule.

When all rules have been examined for two employees and there is no winner, both employees are considered as 'equally suitable'. When there is more than a single 'best' employee in the end, an arbitrary choice is made between the best ones .

Actions to be taken: In this step, we will describe all changes to be made to the schedule of the selected employee. For instance: these could be changes of shifts to resolve the occupancy problem we actually want to fix, but also operations to try to restore the roster quality of the selected employee (e.g. allocation of an additional day off when he obtained too many workshifts or trying to restore a long period of consecutive workdays).

Instead of selecting an employee for an action, the same system is used when a shift or shift change for an employee needs to be selected out of a set of possibilities (in special circumstances).

### 6.2 An order for repairing rosters

Since some problems are easier to repair than others, we should define an order for resolving problems by heuristics. We will first declare the repair order and motivate it in section 6.2.1.
The algorithm will perform the following order of repair steps:

1. Satisfaction of the introduced hard repair shift requirements.
2. Changing the number of workdays for an employee in case more or less days should be assigned than initially provided during the ILP phase.
3. Resolving shortnesses in night shifts during weekends.
4. Resolving shortnesses in night shifts on Wednesday and Thursday.
5. Resolving shortnesses in night shifts on Monday and Tuesday.
6. Resolving shortnesses in late shifts during weekends.
7. Resolving shortnesses in day shifts during weekends.
8. Resolving shortnesses in late shifts during weekdays.
9. Resolving shortnesses in day shifts during weekdays.
10. Optional: try to allocate quality days for SVK-qualified employees.
11. Optional: Provide additional workdays to the flexible employees with a small average deployment ( $2-2.5$ days per week, but this could probably better be performed manually).
12. Optional: try to satisfy collaboration requests.

The scheduling employee could of course perform some additional changes after the algorithm is finished.

For each step in which a shortness is resolved, the kind of shortness may differ. It may be that there is a shortness for a particular type of qualification (and lower types) or that only the total number of assigned employees is too low. Since higher qualified employees may replace lower qualified ones, solving the highest qualification problem may also solve a lower qualification problem (e.g. assigning a Medium Care nurse not only solves an MC shortness problem, but also a problem with the total number of employees that should work in the particular shift).

Therefore, for each step, all shortnesses for the highest qualification level will be resolved first, and then the shortnesses on lower qualifications. For clarity, the repair order during a shortness repair step will be:

1. Senior Nurse (SVK) shortness.
2. Medium Care Nurse (MC) shortness.
3. Basic or Medium Care Nurse (MC or VK) shortness.
4. Shortness in the total number of assigned employees.

After each step, a post check will be performed in which is checked whether all employees still perform the correct number of workshifts. This is more a kind of safety check, since every phase should solve its own introduced problems.
During a step of the algorithm, it is allowed to solve a problem by introducing a new problem which can be resolved in a future step (a step with higher number in the list). On the other hand, it is forbidden to introduce a problem which cannot be resolved anymore. Since all later steps are presumably easier to solve or do not have to be solved at all (e.g. due to the default overflow during day shifts), this will likely not introduce any new problems.
In case the available number of Senior Nurses is low, it might be better to solve SVK shortness in the day shifts before solving all other types of shortnesses. In this way, there is a higher chance more Senior Nurse employees will be assigned to the day shifts. Nevertheless, in the UMC case, the scheduling employee will mostly agree with only one SVK-employee in such situations.

### 6.2.1 Motivation for the repair order

It is important to realize the hard repair constraints before repairing any employee shortness in shifts. Realizing a repair constraint may involve an (additional) shortness in a particular shift, which has to be fixed thereafter. In practice, the number of additional repair constraints is likely to be limited.

It turned out that almost all the occupancy demands in the schedule are already satisfied after the ILP phase. Shortness will particularly arise in night shifts, especially during weekends, due to all preferences and constraints posed by the employees. Since a lot of night shifts are currently assigned to a single employee who performs 21 night shifts per 6 weeks, the total number of night shift shortnesses is still quite small.

Solving a night shift problem is difficult, since a night shift has some special additional constraints (see 1.4.1). To summarize them:

- A night shift period has default start and end days.
- After a night shift period, an employee has a period of at least two consecutive days off.
- Most employees may only perform at most 4 night shifts during a rostering period of 6 weeks. Hence the group of employees who could easily be assigned to an additional night shift is not that large.
- As stated in section 6.1, it is difficult to assign an employee with a weekend off to a work shift during that weekend afterwards. Therefore, the algorithm will not consider that kind of operations.
- When an employee obtains a night shift on Wednesday and Thursday, he must have a day off at both Friday, Saturday and Sunday (since an employee performs the same shift on both weekend days). An employee who has a work weekend can not be assigned to these night shifts, since the weekend can not be changed in a weekend off.

Hence, allocating night shifts during the weekends is by far the most difficult. This means that such problems have to be resolved first, followed by a night shift problem on Wednesday and Thursday (an employee must have a weekend off during the weekend thereafter). Then, night shifts during Mondays and Tuesdays have to be resolved, since an employee has an upper limit in the allowed number of night shifts per schedule.

Subsequently, solving shortness during weekend shifts is more problematic than solving shortness during week shifts. Since surplus is already assigned to day shifts as much as possible (and preferably during week shifts), problems with late shifts should be resolved first.

Also quality days are important to schedule, but resolving problems with the occupancy demands is definitely more important. As already mentioned, there are currently not enough SVK-qualified
employees, so it is not possible to resolve this problem well. The scheduling employee thus has to introduce an ad hoc solution.

### 6.3 Auxiliary functions for repairing

The upcoming sections (starting with section 6.4) will present the heuristics we have developed in order to resolve shortnesses in shifts, introduce hard repair requirements and introduce other desired properties. These heuristics may sometimes introduce other undesired situations, such as employees having too many workdays, having too few days off or having a long period of consecutive workdays.
This section will therefore introduce some auxiliary functions which are regularly needed and mainly used to resolve newly introduced problems in the schedules of individual employees. We will present the following functions to restore a particular introduced problem in the roster of a single employee:

1. Provide an additional day off for a single employee (section 6.3.1). This is needed when a heuristic has introduced an extra workday somewhere in the roster. It is in general always possible to successfully apply this function (see the note in the corresponding subsection).
2. Provide an additional workday for a single employee (section 6.3.2). This is needed when a heuristic has introduced an extra day off somewhere in the roster. It is always possible to successfully apply this function.
3. Resolve a long work period in the roster of a single employee (section 6.3.3). This may be needed when a heuristic has introduced an extra workday somewhere in the roster, whereby a long period of consecutive workdays is created. Unfortunately, this function can only resolve such problems when it can be done without introducing other major problems (such as the introduction of unresolvable shortnesses).
4. Resolving an introduced Late $\rightarrow$ Day transition in the roster of a single employee (section 6.3.4).

We will also introduce another auxiliary function which does not consider the roster of a single employee:
5. Try to allocate a shift to a flexible employee (section 6.3.5).

### 6.3.1 Schedule a day off (function: ProvideDayOff())

Provide an additional day off to a particular employee $j$ in case he obtained too many workdays.

### 6.3.1.1 Preselection

Select all possible workdays to change in a day off:

1. The employee performs a workshift on the particular day.
2. It is not a weekend day.
3. The employee does not perform a night shift.
4. By changing the shift, no hard permanent/repair requirement will be infringed.
5. We will not introduce a shortness in a shift which cannot be resolved in a later phase.

It is in general always possible to successfully apply this function, since the number of available employees is large. Therefore, it is likely possible to change a day shift into a day off for this employee. In the unlikely situation that there is no remaining possibility, this problem has to be resolved manually.

### 6.3.1.2 Sorting (the preselected workdays)

1. The employee performs a shift for which a retraction will not introduce a shortness.
2. The day falls within a long period of consecutive workdays (e.g. at least $a_{j}+1$ ).
3. The day falls within the longest work period of the employee.
4. The workday is immediately followed or preceded by a day off.
5. The workshift to retract has an overflow in the number of allocated employees and the extend of overflow is bigger.
6. The employee performs a late shift.

### 6.3.1.3 Actions to be taken

1. Change the workshift of the employee in a day off.

### 6.3.2 Schedule a workday (function: ProvideWorkDay())

Change a day off for a particular employee $j$ into a day or late shift. It is initially allowed to introduce a Late $\rightarrow$ Day transition, but we will try to resolve it immediately thereafter (see subsection 6.3.4 about repairing Late $\rightarrow$ Day transitions).
In case a shift shortness is introduced to resolve the Late $\rightarrow$ Day transition, it must be possible to resolve this shortness in a later phase. Furthermore, resolving a Late $\rightarrow$ Day transition in this situation cannot be performed by inserting a day off, since this will undo the effect of inserting a workday.

### 6.3.2.1 Preselection

Create a set of unique tuples. Each tuple consists of a current day off and a day or late shift. All of them have to satisfy the following demands:

1. It is not a weekend day.
2. The employee has a day off.
3. There is no hard permanent/repair requirement for a day off on this day.
4. The day is not within 2 days after a night shift period.
5. By scheduling the work shift, the employee obtains at most $\left\{\min \left(6, w_{j}+1\right)\right\}$ consecutive workshifts.
6. The shift to allocate is not forbidden by a hard permanent/repair requirement.
7. When a Late $\rightarrow$ Day transition is introduced, and introducing a shift shortness is needed to resolve it, it must be possible to resolve the shortness in a later phase. It is not allowed to resolve the Late $\rightarrow$ Day problem by allocating a day off.

### 6.3.2.2 Sorting

1. An introduced Late $\rightarrow$ Day transition could be resolved without any negative consequences for the shift occupancy demands.
2. The action will not lead to a freestanding day off (a 'free shift'-block of only a single day).
3. A shortness in the occupancy demands is (partly) resolved by the allocation action.
4. An already existing period of consecutive workdays is extended with one day (no interruption of a free shift block).
5. The current number of workdays during the week of allocation is smaller.
6. A day shift is allocated.

### 6.3.2.3 Actions to be taken

1. Change the day off in the selected work shift.
2. Resolve an introduced Late $\rightarrow$ Day problem by performing ResolveLateDay() without introducing a day off (the preselection guarantees this is possible).

### 6.3.3 Resolve a long work period (function: ResolveLongWorkPeriod())

Sometimes, an (introduced) long period of consecutive workdays has to be changed. This is done by inserting a day off somewhere in the period, but only when no shortnesses are introduced which cannot be resolved in a later stage. Otherwise, the problem has to be resolved manually.

The function ProvideDayOff() is only applied on the block of workdays. When needed, the employee obtains a new workday afterwards using the function ProvideWorkDay().

### 6.3.4 Resolve a Late $\rightarrow$ Day transition (function: ResolveLateDay())

This function is used when a repair heuristic inserts a shift $s$ on a particular day $d$ and introduces an undesired Late $\rightarrow$ Day transition. We cannot change this shift into a day off, since we will then undo the previous operation. When the inserted shift is a day shift, the problem has to be resolved by changing some preceding late shifts, otherwise it has to be resolved by changing some upcoming day shifts.
We can try to change the preceding or the next shift in the same shift as the introduced one, but in this way it might be that we shift the problem to another day in the roster. If this process is repeated a few times, a substantial number of shifts will be changed before the problem is finally resolved. We will therefore try to break this problem shifting by inserting a day off as soon as possible.

- Let $d$ be the day of the inserted shift and $s$ the inserted shift.
- In case $s$ is a day shift: Let $s^{\prime}$ be a late shift and let $d^{\prime} \leftarrow d-1$.
- In case $s$ is a late shift: Let $s^{\prime}$ be a day shift and let $d^{\prime} \leftarrow d+1$.

As long as the problem has not been resolved: try to execute the steps below. The value of $d^{\prime}$ may be updated during this process. In case no step can be applied, it will be impossible to resolve the problem.

1. In case shift $s^{\prime}$ is not assigned on day $d^{\prime}$, the problem has been resolved. STOP.
2. Try to allocate a day off on day $d^{\prime}$, in case:

- No hard (permanent/repair) shift requirement is infringed.
- Day $d^{\prime}$ is not during a weekend.
- There is no shortness introduced in the original shift $s^{\prime}$ on this day, which could not be resolved during a later phase of the repair algorithm.
When this action was successful, the problem has been resolved. STOP.

3. Try to change the shift on day $d^{\prime}$ into shift $s$, in case:

- No hard (permanent/repair) shift requirement is infringed.
- There is no shortness introduced in the original shift $s^{\prime}$ on this day, which could not be resolved during a later phase of the repair algorithm.

When this action was successful, update $d^{\prime}$ and repeat the routine, starting at step 1.

- In case $s$ is a day shift: Let $d^{\prime} \leftarrow d^{\prime}-1$.
- In case $s$ is a late shift: Let $d^{\prime} \leftarrow d^{\prime}+1$.

4. There is no solution possible when none of the previous steps could be applied.

In case the problem has been resolved, we will also check whether it is needed to assign an additional workday, using ProvideWorkDay().

### 6.3.5 Allocate a shift to a flexible employee (function: AssignFlexEmp())

This function tries to allocate a day or late shift with shortness (in a certain qualification or in the total number of employees) to a flexible employee. When the employee is assigned to a weekend shift, he will retrieve the same shift on the other weekend day. Hence, during weekends, all checks have to be performed double.

- Let $a_{j}^{\prime}$ be the average number of days per week a flexible employee $j$ is deployed normally.
- Let $l_{j}$ be the number of already allocated workdays for a flexible employee $j$ for this rostering period of $k$ weeks $(k=6)$.
- Let $p_{j}=\frac{l_{j}}{k \cdot a_{j}^{\prime}}$ be the percentage of deployment for flexible employee $j$ for this rostering period.
- Let $\tau$ be the number of already allocated workweekends for flexible employee $j$ for this rostering period.

First, all flexible employees which were not already scheduled during the ILP phase are selected. Hence, we select all flexible employees who:

- Currently have a schedule with only days off.
- Already have some shifts assigned as a result of using AssignFlexEmp() in a previous repair situation.

Then, this group is filtered in the following way:

1. The employee can resolve the shortness problem.
2. The employee may obtain the shift on the particular day.
3. In case of a weekend: the employee still does not have 2 work weekends $\left(\tau_{j}<2\right) .^{3}$
4. We do not introduce a Late $\rightarrow$ Day problem by allocating the shift.
5. The employee does not get assigned to more than $\left\lceil a_{j}^{\prime}+1\right\rceil$ shifts during the particular week (note that two shifts will be allocated during a weekend).
6. The employee does not obtain more than $\left\lceil k \cdot a_{j}^{\prime}+1\right\rceil$ shifts during the full rostering period of $k$ weeks (again: special attention during weekends).

In case no employee remains after the filtering actions, it is not possible to resolve the shortness problem by allocating a flexible employee. In case more employees remain, the one with the smallest percentage of deployment (smallest $p_{j}$ ) will retrieve the shift (or shifts in case of a weekend).

[^22]
### 6.4 Allocation of a hard repair requirement

This section will describe how a hard repair requirement is scheduled. First, we will repeat that it is not possible to schedule:

- A night shift (period)
- A work shift during a weekend off.
- A day off during a workweekend.

It is however possible to schedule:

- A day shift or late shift on a weekday.
- A day shift or late shift on a weekend day, provided that the weekend is not off.
- A day off on a weekday.

When an employee wants a special regulation for a weekend, it has to be taken into account in the roster generation phase.

In case a Late $\rightarrow$ Day transition is introduced, it is tried to resolve this problem using the function ResolveLateDay(). When there is no possibility to overcome this, then this is because of conflicting hard requirements. The algorithm will then keep the newly introduced Late $\rightarrow$ Day transition and the problem has to be resolved manually.

This section is divided in two parts:

1. We will firstly discuss a simple operation which can be used when the hard requirement does not override a night shift period.
2. Although we will not override existing night shift periods, we will provide a more complex solution for situations where a night shift period must be overridden, just for the sake of completeness. We will also introduce an order for hard requirement allocation in case the retraction of a night shift period is allowed.

### 6.4.1 The basic algorithm for scheduling hard repair constraints

This operation can be used when the hard requirement does not override a night shift period. It could be that we introduce a shortage of employees in a particular shift, but this can be resolved later (see the repair order in section 6.2)

### 6.4.1.1 Scheduling a day off on day $d$

1. Simply change the original shift on day $d$ in a day off.

### 6.4.1.2 Scheduling a day or late shift on day $d$

In case $d$ is a day during a weekend which is currently allocated off, the action cannot be performed.

1. Assign the day or late shift on day $d$
2. In case $d$ is a weekend day, also assign this shift on the other weekend day
3. Try to resolve a potential Late $\rightarrow$ Day-problem, using ResolveLateDay().

### 6.4.1.3 Post-processing

After each action:

- Resolve an introduced long work period when needed, using ResolveLongWorkPeriod().
- Allocate some additional workdays when needed, using ProvideWorkDay().
- Allocate some additional days off when needed, using ProvideDayOff().


### 6.4.2 An extended algorithm for scheduling hard repair constraints

Although we will not override existing night shift periods, this section provides a more complex solution for situations where a night shift period must be overridden, just for the sake of completeness.
Overriding a night shift period is needed in case:

- A night shift is overridden directly.
- The buffer of two consecutive days off after a night period is overridden.

When retraction of a night shift period is allowed, the total repair operation becomes a little bit more complex.

The night shift block consists of $2-4$ consecutive night shifts, followed by at least two days off. If we are lucky, only 2 shifts have to be changed, but it could also be 4 . Since this is a 'large' piece of the total roster, we may easily introduce an unbalance in an employee's roster. There is a big chance a lot of auxiliary functions (section 6.3) will have to be applied in order to try to resolve these unbalances. Hence, the overall quality of the employee's roster will likely drop considerably.
Initially, we wanted to create a solution in which the roster remains somewhat balanced, without introducing Late $\rightarrow$ Day problems and overriding any of the employee's hard permanent constraints (of the type 'the employee must perform shift $x$ '). The original idea was therefore to perform the night shift block retraction and hard constraint allocation as one single operation. Therefore, we tried to analyze all different combinations of night shift blocks (weekend, small block during weekdays, large block during weekdays), hard repair shift requirements (inserting day and late shifts and days off), and existing hard permanent shift requirements as a first approach. Of course the many different operations became very complex.
In the end, this approach did not work out well, due to its high complexity. Therefore, we first decided to provide some overall suggestions of the complex method about how the problem could be handled. Nevertheless, since the content of this chapter has been refined multiple times, a lot of repair heuristics slowly became more and more combinations of small operations instead of a single big operation. Then suddenly, a quite simple approach appeared to be possible, although we do not expect very good results.

### 6.4.2.1 The general approach of night shift block retraction and shift allocation

The general idea is that the allocation of the hard repair requirement and the retraction of the night shift block will not be performed in a single operation, but will become a composite operation.

1. First, we will retract the night shift block and replace it with some other possible shift combination. We do not care whether the hard repair constraint to insert will be satisfied (although it can be taken into account). The only thing we are concerned with is a valid retraction of the night shift period.
2. Thereafter, we will assign the hard repair constraint to the employee, if it was not already satisfied during the first step. Since a day or late shift or a day off will now be allocated, the solution can be found in section 6.4.1.

### 6.4.2.2 Retracting a block of night shifts

First, create all possible permutations of day and late shifts and days off for the particular workdays. Since a night shift block consists of at most 4 days and there are 3 possible shift allocations for each day, the number of possible permutations is at most $3^{4}=81$. Thereafter, all illegal permutations will be filtered (preselection) and the remaining ones will be sorted to find the best solution. We will work this out below.

## Preselection

As stated in section 6.1: in case a night shift period has to be retracted, a hard permanent requirement of the type 'do not allocate...' may be overridden. A permutation is therefore possible when:

1. It does not infringe any hard permanent/repair requirements (except the not requirements).
2. A workweekend is not changed in a weekend off.

## Sorting

All permutations are ordered to find the best suitable one. We will do this in the following manner:

1. The permutation does not overrule a hard not-requirement.
2. The permutation does not introduce a Late $\rightarrow$ Day problem within the old night shift period (i.e. the permutation itself has a day shift after a late shift).
3. The permutation does not introduce a Late $\rightarrow$ Day problem on the boundary of the old night shift period (i.e. the permutation starts with a day shift, while the last day before the night shift period was a late shift).
4. The permutation already satisfies the hard repair requirement to insert (only when a night shift period is overridden directly).
5. The permutation allocates more workdays (than another permutation). In this way, it is tried to remain a good spreading of workdays and days off.
6. The permutation does not introduce a single day off.
7. The permutation does not introduce a single workday.
8. The permutation allocates more day shifts (since surplus has to be put into the day shifts).

In this way, hard not-requirements are tried to be kept, and the illegal Late $\rightarrow$ Day situations are tried to be prevented (but we allow them in case we do not have another option). Furthermore, it is tried to keep the employee's roster balanced and to not allocate too many late shifts.

## Actions to be taken

The night shift period is replaced with the shift permutation. In case a Late $\rightarrow$ Day problem is introduced on the boundary of the old night shift period (see above), it is tried to solve this with ResolveLateDay().

Thereafter, the hard repair constraint is realized as described in section 6.4 .1 (in case it is still needed).

### 6.4.3 An order for hard requirement allocation

When we allow night shifts to be retracted, we could create an order for allocation of hard repair constraints. A reason for doing this, is again trying to keep the roster quality for an employee more acceptable.
As already said, the roster quality may drop substantially when a large block of shifts is changed from the initial situation and it is likely that some surplus in the newly allocated shifts is introduced, since for most days all occupancy demands are already satisfied after the ILP phase.

When a hard repair constraint is allocated, most of the times an auxiliary function (section 6.3) must be applied afterwards in order to resolve an introduced undesirability. When an extra workday or day off should be provided by a recovery utility, a retracted night shift period may be an easy place in the roster on which to apply the recovery operation, because it is likely that the employee is surplus in the allocated workshifts on these days.

Furthermore, operations without retraction of a night shift block may have a smaller impact on the roster quality and although a recovery operation (such as allocation of another workday or day off) may be needed after a simple change, they may be easier to perform.

Therefore, it might be better to perform necessary night shift block retractions as soon as possible.
The hard requirements will thus be scheduled in the following order:

1. All requirements for which a night shift block has to be indirectly retracted. This is the case when the requirement considers a day on which a day off was allocated, which is within two days after a night shift block (the buffer period). Not only the night shift block must be changed, but also a part of the buffer period thereafter.
2. All requirements for which a night shift block has to be directly retracted. This is the case when the requirement considers a day on which a night shift was allocated. It is likely that we do not have to change the buffer period.
3. All other requirements.

### 6.5 Resolving night shift shortages

Resolving an occupancy problem during a night shift period is initially performed by shifting the problem to the day and late shifts of the corresponding days.

We distinguish three major types of night shifts blocks:

- Night shifts on Monday and Tuesday.
- Night shifts on Wednesday and Thursday.
- Night shifts during a weekend (Friday, Saturday and Sunday).

Although an employee could have a night shift block from Monday until Thursday, this is not a major type, since it could be that a part of this block does not have occupancy problems.

We would like to note that we have defined a repair order for night shift problems in section 6.2. When the night shift problem occurs during a weekend, it is only possible to allocate an employee for the night shift period if he already has a workweekend. On the other hand, when it occurs during a Wednesday and Thursday, an employee must have the subsequent weekend off, since at least the Friday and Saturday should be a day off (a buffer of at least two days off).
This section is divided in two parts:

1. We will firstly discuss the basic heuristics to resolve shortness during a night shift period.
2. The second part of the section will provide some unimplemented suggestions for more complex heuristics.

### 6.5.1 Basic heuristics to resolve night shift shortness

We will now provide the 'preselection', 'sorting' and 'actions to be taken' steps (as defined in section 6.1.4). The latter steps are equal for all three types of night shift blocks. Only the 'preselection' step is different, but the situations are quite similar to each other.

### 6.5.1.1 Preselection

Select all employees who satisfy all following rules:

## Resolving night shift problems during the weekend

1. The employee has a day shift or late shift during the weekend.
2. An illegal night shift period is not introduced (e.g. Wednesday - Sunday).
3. Another night shift period is not overridden (e.g. the next Monday and Tuesday are changed in days off).
4. The employee does not have a hard permanent/repair requirement (see section 6.1.1.2) which forbids a night shift on Friday, Saturday or Sunday or a day off on Monday or Tuesday.
5. The employee is allowed to perform night shifts (no students and flexible employees).
6. The employee currently has at most $\eta_{j}-3$ allocated night shifts during this rostering period.

## Resolving night shift problems on Wednesday en Thursday

1. The employee has the next weekend off.
2. The employee does not have a hard permanent/repair requirement which forbids a night shift on Wednesday or Thursday and a day off on Friday.
3. The employee is allowed to perform night shifts (no students and flexible employees).
4. The employee currently has at most $\eta_{j}-2$ allocated night shifts during this rostering period.

## Resolving night shift problems on Monday and Tuesday

1. The employee does not have a hard permanent/repair requirement which forbids a night shift on Monday or Tuesday.
2. The employee does not have a hard permanent/repair requirement which forbids a day off on Wednesday or Thursday OR there already is a night shift assigned on both Wednesday and Thursday.
3. An illegal night shift period is not introduced (e.g. Friday - Tuesday).
4. The employee is allowed to perform night shifts (no students and flexible employees).
5. The employee currently has at most $\eta_{j}-2$ allocated night shifts during this rostering period.

### 6.5.1.2 Sorting

1. The employee does not have the SVK qualification (we definitely need them for the day shifts).
2. The employee does not have a night shift block during the previous or upcoming 7 days (except when a block during weekdays could be extended).
3. The employee has the smallest number of allocated night shifts.
4. The employee introduces less shortnesses in his currently allocated shifts on all days for which the shift has to be changed.
5. The employee already has a workshift on all days which have to be changed in a night shift.
6. The employee already has a workshift on some days which have to be changed in a night shift.
7. The employee already has two days off on the days after the night shift block to insert.
8. The employee already has at least one day off on the days after the night shift block to be inserted.

### 6.5.1.3 Actions to be taken

1. Provide the employee the night shift block.
2. Provide the employee two days off after the night shift block.
3. Resolve an introduced long work period when needed, using ResolveLongWorkPeriod().
4. Allocate some additional workdays when needed, using ProvideWorkDay().
5. Allocate some additional days off when needed, using ProvideDayOff().

### 6.5.2 Shifting the problem to another night shift period (suggestion)

In case it is impossible to allocate an employee for a night shift block on a couple of days (for instance because all available employees already have night shifts in another part of the roster and could not obtain more, since there is an upper limit), the problem might be resolved by retracting another single night shift block of an employee. We will not consider retraction of multiple blocks, since the employee's roster will then likely be rather destabilized.

We only suggest - but did not implement - this step.

### 6.5.2.1 Resolving weekend problems by shifting them to weekdays

To solve a weekend problem, we could shift the problem to a night shift period during weekdays. First we can try to retract a night shift block of 2 days for an employee. When this does not help, we could retract a full block of 4 days, but this is in fact not really desired.
Of course we could also try to shift the problem to another weekend, but since night shift problems are already difficult to solve during weekends, it will probably not help that much.

## Preselection

First, select all employees who probably could perform the night shift block (check the earlier defined rules, except the number of already allocated night shifts). Thereafter, create tuples of the selected employees and their exchangeable night shift blocks.

Example 6.2 (Tupling night shift periods). Suppose an employee may perform a night shift period during the weekend. He currently has a night shift period of Monday - Thursday, somewhere in his roster.

Depending on the constraints during the weekdays' night shift period, three tuples may be introduced:

1. The employee, together with the shifts on Monday and Tuesday.
2. The employee, together with the shifts on Wednesday and Thursday.
3. The employee, together with all shifts on Monday to Thursday.

The tuples should satisfy the following additional rules:

1. In case the retractable night shift period consists of 2 days, the employee should currently have at most $\eta_{j}-1$ night shifts.
2. The retractable night shift period is not during the weekend.
3. The retractable night shift period could be changed in some other shifts, without introducing unsolvable Late $\rightarrow$ Day problems and infringing any hard permanent/repair constraints. Hence there must be a possible permutation as defined in section 6.4.2.

## Sorting

It is first tried to exchange a night shift block of only 2 shifts. Besides, a retraction of a block on Monday and Tuesday is more preferred, since repairing it is slightly easier than the Wednesday and Thursday block (because then the employee should have the next weekend off). We will sort the tuples in the following manner:

1. Only a night shift block of 2 days is retracted.
2. The night shift block consists of two shifts on Monday and Tuesday.
3. All normal sorting rules for night shift blocks (see above) can now be executed.

## Actions to be taken

The old night shift block is first retracted, using the method described in section 6.4.2. Thereafter, the new night shift block will be assigned, as described before.

### 6.5.2.2 Resolving problems on Wednesday and Thursday by shifting them

Although the same approach could be used for weekdays, it is more difficult to perform, since we may introduce a cycle of allocating and retracting night shift blocks. Therefore, a tabu list should be introduced to maintain the most recent old states and hence prevent cycling.

## Preselection

Again, select all employees who probably could perform the night shift block (check the earlier defined rules, except the number of already allocated night shifts). Thereafter, create tuples of the selected employees and their exchangeable night shift blocks. The tuples should satisfy the following additional rules:

1. The retractable night shift period is not during the weekend.
2. The retractable night shift period could be changed in some other shifts, without introducing unsolvable Late $\rightarrow$ Day problems and infringing hard permanent/repair constraints.
3. The retractable night shift period is not tabu (just resolved in the past $x$ actions).

## Sorting

Retracting a night shift block on a Monday and Tuesday is more preferred for this step than retracting a block on another Wednesday and Thursday (since a block on Wednesday and Thursday is more difficult to resolve). Hence we will sort the tuples in the following manner:

1. The night shift block consists of two shifts on Monday and Tuesday.
2. All normal sorting rules for night shift blocks (see above) can now be executed.

## Actions to be taken

The actions to be taken are equal to the weekend situation.

## Tabu search length

In case the problem has not been fixed after a small couple of problem shifts (since the individual schedules will otherwise become very bad), we will not try another and mark the problem as unresolvable.

### 6.5.2.3 Resolving problems on Monday and Tuesday by shifting them

This case is equal to the case on Wednesday and Tuesday, except that only an exchange of another night shift period on Monday and Tuesday is considered.

### 6.5.2.4 Final notes on shifting night shift problems

Since the method has not been implemented, we cannot tell whether the results will be any good. However, we have tested almost all other heuristics as described in this chapter and therefore we could certainly make a prediction.

As mentioned in the final part of this chapter, too many change actions could easily lead to bad individual roster quality. Since a night shift period (including the days off thereafter) is a substantial chunk of the roster, retracting it may introduce some unbalance, certainly when an undesired (low in the sort hierarchy) action is chosen. Performing such actions a few times in a tabu search manner may likely drop the roster quality tremendously.

Besides, since experience shows that implementing good heuristics can be quite difficult, this may certainly hold for the implementation of shifting night shift problems.

### 6.6 Resolving day and late shift shortages

This section will describe how an occupancy problem in a day or late shift is resolved.
Since almost all surplus during weekdays and weekends is assigned to the day shifts, we can use these shifts as a buffer to solve problems during late shifts. Furthermore, the ILP phase will likely not have introduced any shortness during a day shift (except for the senior nurse (SVK) demand), so we only expect a few problems with late shifts.

For a weekend shift, it is first tried to schedule a flexible employee. In case this is not possible, a problem with a late shift will always be shifted to the day shift. For a day shift, the algorithm will check whether there is an employee surplus during the late shift, but there is only a small chance this will occur. It will never be the case that the problem is shifted back to the late shift. Otherwise, the problem has to be resolved manually.

The weekday method is comparable to the weekend method. The most important difference is the additional possibility to change a day off for an employee in a workday.

Also this time we will provide the 'preselection', 'sorting' and 'actions to be taken' steps (as defined in section 6.1.4). In case the expression Weekend or Weekday is written, the corresponding rule will only be used while resolving problems during the concerning period of the week.

### 6.6.1 Preselection

Select all employees who:

1. Weekend: Do not have a weekend off.
2. Do not already perform a night shift or the concerned shift.
3. Do not have a hard (permanent/repair) constraint which forbids an allocation of the concerned shift (check both weekend days).
4. Do not introduce a Late $\rightarrow$ Day problem which cannot be resolved.

### 6.6.2 Sorting

The sort function firstly tries to introduce as little additional problems as possible. Thereafter, it tries to keep a good overall quality of the employee's schedule.

We distinguish slightly different functions for repairing late shifts and repairing day shifts.

### 6.6.2.1 Repairing late shifts

1. The employee does not have the SVK qualification (we definitely need them for the day shifts).
2. The employee does not introduce a shortness in another shift, due to the action.
3. When two employees are equal: check the rules as described under day shifts.

### 6.6.2.2 Repairing day shifts

1. The employee does not obtain a period of too many consecutive workdays.
2. Weekday: The employee does not have a day off on the particular day.
3. The employee does not introduce a Late $\rightarrow$ Day problem.
4. The number of late shifts becomes more fairly distributed. ${ }^{4}$

### 6.6.3 Actions to be taken

For all situations, the following actions have to be taken:

1. Provide the employee the shift (day or late). For a weekend: on both weekend days.
2. Resolve a Late $\rightarrow$ Day-problem when needed, using ResolveLateDay().
3. Resolve an introduced long work period when needed, using ResolveLongWorkPeriod().
4. Allocate some additional workdays when needed, using ProvideWorkDay().
5. Allocate some additional days off when needed, using ProvideDayOff ().

### 6.7 Possible extensions

This section will describe some other problems which can (partly) be resolved afterwards using heuristics.

[^23]
### 6.7.1 Scheduling of flexible employees

A flexible employee is an employee whose deployment is requested, once needed. Therefore, it is normally unnecessary to assign these employees to more shifts than actually needed, but the UMC prefers to do so. However, we have chosen not to implement this step, since the scheduling employee will likely want to make some manual changes to the resulting roster after the repair phase. While performing these manual changes, it will be handy when there are remaining flexible employees who can be deployed. Thereafter, the scheduling employee could assign them to a few more shifts at his own discretion.

For the sake of completeness, this section will (shortly) discuss two strategies for automatically scheduling a flexible employee for a roster period of $k(=6)$ weeks. We assume a flexible employee will work at most 3 days a week on average. A flexible employee will not retrieve night shifts.

### 6.7.1.1 Notation

We will use the following additional notation in this section:

- Let $a_{j}$ be the average number of workdays per week for flexible employee $j$.
- Let $a_{j}^{\prime}=\left\lfloor a_{j}\right\rfloor$ be the number of workdays we will definitely provide each week (whenever possible).
- Let $r_{j}=k \cdot\left(a_{j}-a_{j}^{\prime}\right)$ be the remainder.
- Let $\gamma$ be a measure of undesirability for an employee surplus during a shift. We will define: $\gamma_{\text {day }}=1$ and $\gamma_{\text {late }}=2$, hence each surplus during a late shift is two times as bad as each surplus during a day shift.


### 6.7.1.2 The greedy strategy

The first strategy is very simple, but also very greedy. Each flexible employee $j$ will be heuristically scheduled, starting from the first week:

1. Create sets of $t=a_{j}^{\prime}$ distinct work shifts for the considered week. There are $2 \times 7=14$ possible work shifts (day or late) to choose from. The number of sets is small and all sets are enumerable.
2. Remove all sets in which:

- An already scheduled workshift is not selected.
- A hard permanent / repair requirement is infringed.
- Only half of the weekend is allocated.
- A Late $\rightarrow$ Day problem is introduced.
- A day shift is allocated on Monday, while a late shift was allocated on the previous Sunday.
- A period of more than $t+1$ consecutive workdays will arise (as a result of the allocation for the previous week or already allocated shifts for the next week).

3. In case there are no sets left, try again for $t \leftarrow t-1$ as long as $t>0$. Otherwise, the employee is not scheduled at all (a likely reason is that the employee is having a holiday during that week).
4. The employee obtains a set of workshifts for which the total employee surplus in all allocated shifts is the lowest. The total employee surplus of a shift is defined as $\gamma(\beta-\alpha)$, where $\beta$ is the total number of assigned employees after assigning the flexible employee and $\alpha$ is the desired maximum number of employees.

In case $r_{j}>0$, then try to allocate $r_{j}$ extra shifts by providing an extra shift for each week in which there are $t$ shifts scheduled. This is done by at most $r_{j}$ times applying the strategy above with $t=a_{j}^{\prime}+1$.

### 6.7.1.3 The ILP strategy

We could also create an ILP in order to assign the flexible employees. We will not work out all the details, but we will provide some pointers.

It is possible to enumerate all possible weekly schedules for all flexible employees as defined in section 6.7.1.2 above. We could then compose an ILP formulation which tries to select a schedule for every employee for every week, such that the amount of employee surplus is leveled over the day (and late, with some kind of penalty) shifts.
Another approach is the following one, which tries to schedule directly for $k$ weeks using an (I)LP, which is solved by using column generation. We assume at most $k=6$ weeks are considered.

We could generate sets of schedules for every week. A set $S_{j}^{w}$ contains all possible schedules consisting of $a_{j}^{\prime}$ or $a_{j}^{\prime}+1$ (probably consecutive) shifts - for flexible employee $j$ for week $w$. In case a set is empty, we try to create it by allocating less shifts, until we have obtained some possible schedules.

Thereafter, all sets for an employee $j$ are added into a layered graph $G_{j}$. The graph consists of $k+2$ layers. Layer 0 contains a single node $s_{j}$ (the source) and layer $k+1$ contains a single node $t_{j}$ (the sink). A layer $w=1 \ldots k$ contains nodes for all schedules in the set $S_{j}^{w}$. All nodes in a layer $w$ are connected to all nodes in the layers $w-1$ and $w+1$. The edge between a node $u$ and $v$ obtains a cost $c_{u v}$ which reflects how well schedule $v$ succeeds schedule $u$. This cost is based on the number of consecutive workdays, introducing a Late $\rightarrow$ Day transition (high penalty), etc.

Furthermore, we could also assign a cost to each schedule (which can be added to all outgoing edges). This cost can for instance reflect whether the schedule contains a workweekend in order to provide as little workweekends as possible.

A path from $s_{j}$ to $t_{j}$ in a graph $G_{j}$ will now represent a full schedule of $k$ weeks for flexible employee $j$. Of course the shortest path represents the best possible schedule for the employee, although it could be better to select another schedule in order to minimize the number of employees surplus in all shifts. We can use these paths to select the best columns in a column generation approach. The costs of the edges can be adjusted to reflect the reduced costs of a column (see also section 2.3.2).

### 6.7.2 Allocation of quality days for senior nurses (SVK)

Since the number of available senior nurses is currently quite low, it will be difficult to allocate quality days. In the current UMC situation, an ad-hoc solution by the scheduling employee may be better to perform. Section 5.2.3 introduced an ILP-solution for this problem. This section will provide two suggestions for the allocation of quality days using heuristics.
Using the following two sets of actions, it is tried to have more senior nurses during the day shift on a weekday $d$.

### 6.7.2.1 Move a redundant senior nurse to the day shift

In case there is a senior nurse assigned to the late shift on day $d$ and he could obtain the day shift on the same day, we can perform an easy action by shifting this employee to the day shift. However, it is unlikely that this situation will actually occur. We could also try to apply an initial step first: try to assign a flexible employee to the late shift on day $d$ first, such that the senior nurse becomes surplus.

In detail:

1. Find a day $d$ with a surplus in the late shift, such that an allocated senior nurse can be retracted from the late shift and assigned to the day shift without introducing any qualification shortness in the late shift. In case this is not possible, we could check whether we can assign a flexible employee $y$ to the late shift, using the function AssignFlexEmp (). This employee $y$ should have all qualifications for which there is a shortness when employee $x$ is retracted from the shift.
2. Since we do not want to introduce a Late $\rightarrow$ Day transition for the senior nurse, the employee must also have a day shift or day off on day $d-1$. We may only introduce a Late $\rightarrow$ Day transition in case we are able to resolve it (using ResolveLateDay()) without introducing additional shortnesses in a shift.
3. Change the shift allocation of the employee in a day shift.

### 6.7.2.2 Swap the shifts of a senior nurse and another employee

A more likely situation is one in which a senior nurse can swap his late shift with another employee who performs a day shift on the same day.

1. Find a senior nurse $x$ with a late shift on day $d$, for whom a Late $\rightarrow$ Day problem will not be introduced (or can be resolved without introducing additional shortnesses) when the shift is changed in a day shift.
2. Check whether an employee $y$ who performs a day shift on day $d$ could be allocated to the late shift.

- We should verify that we do not introduce a Late $\rightarrow$ Day problem for employee $y$ (he does not have a day shift on day $d+1$ ).
- Employee $y$ should have all qualifications for which there is a shortness when employee $x$ is retracted from the shift.

3. In case there is a possibility: swap the shift allocation of the employees.

### 6.7.3 Student - supervisor collaboration

This problem is also treated in section 5.2.4. One of the conclusions was that such requests might already be satisfied after solving the ILP, in case it is only needed to have a predetermined fraction of overlapping shifts. We can also use some heuristics to satisfy such constraints.
Most collaboration issues will occur between normal employees and students. Since students are largely allocated to day shifts as surplus, it will not be a real problem to change their shift allocation on some days. Hence, we could try to change the workshifts of the student on some days into the same shift as the supervisor, but only when this does not introduce any shortness in a shift.

We will first try to perform such steps without introducing a Late $\rightarrow$ Day problem. Otherwise, we may introduce it, as long as it could be resolved (ResolveLateDay()) without introducing shortness in another shift.

The general situation (two employees $x$ and $y$ who should collaborate) is more difficult in case such a simple change is not possible. We could perform a similar approach as the allocation of quality days (section 6.7.2). For all days on which both employee $x$ and $y$ perform a day or late shift (but not the same), we will check whether one of them could exchange his shift with another employee $z$.

We will first consider simple exchanges (without any issues which must be resolved) of day and late shifts. Thereafter, more complex exchanges which introduce recovery utilities (especially of Late $\rightarrow$ Day problems) might be tried.

We might also consider a change of some days off into a workshift (whenever possible), but we suppose the overall roster quality of the employee will then drop considerably. Hence, we will not advise to perform such steps.

### 6.7.4 Final notes on extensions

The repair heuristic (and the associated recovery utilities) is a powerful tool to resolve scheduling problems after the ILP solution has been calculated. It can be used for a wide variety of problems and more kinds of extensions could be thought of. However, we must be careful when it is used, since the quality of the individual rosters may drop tremendously.

Whenever possible, it is always more preferred to introduce a special demand in the ILP or generation phase than resolving it afterwards.

### 6.8 Experiments and results

We have implemented a large part of the proposed heuristics. We are able to insert hard repair shift requirements, we could fix night shift periods and day or late shift periods. Furthermore, we could change the number of allocated workshifts for an employee and we have implemented all recovery utilities. However, we will never consider the retraction of a night shift period.

In case a shortness during a day or late shift must be resolved, we will first try to allocate the shift to a flexible employee, as described. However, we will not provide full schedules for these employees, since they can be used easily when needed during a manual completion of the schedule.

We have to be careful which employees will be concerned while repairing, since we have normal employees, employees with very restricted schedules, employees who will not be scheduled at all, flexible employees who have already been scheduled using the DFS-pattern and flexible employees who have not been scheduled yet. We will therefore distinguish 3 groups of employees:

1. Most employees who are scheduled using the ILP. These are all employees with a DFSpattern, a predefined-weeks pattern or a (set of) predefined roster(s) for who repairing is not explicitly forbidden (see the third group for examples). This includes the set of flexible employees that is directly scheduled using the ILP.
2. All remaining flexible employees, who are not already scheduled using the ILP.
3. All other employees. These are the employees who will not be scheduled during the period and the employees with a very special schedule. An example of an employee with a very special schedule is the one with alternately a full week of night shifts and a full week off. Although this is a predefined roster, the pattern of alternating weeks with night shifts and days off is not allowed to be changed.

The employees in the first group will be treated in all heuristics (except the AssignFlexEmp() function). The ones in the second group will only be treated in the AssignFlexEmp() function. The schedules of the remaining employees will not be modified using the set of repair heuristics.

The reason that employees with a predefined-weeks schedule are classified in set 1 is that they may have a night shift period during a predefined part of the week. We should however note that we do not check whether an otherwise illegal pattern is introduced (but we insert some hard permanent shift requirements in order to try to prevent it).

We will perform two kinds of experiments. First, we will perform a normal repair operation on the schedules for the period February - March 2013. Thereafter, we will try to allocate some additional shift requirements in order to test the robustness of the heuristics.

### 6.8.1 Repairing the schedules for February - March 2013

We have automatically repaired the two calculated rosters for February - March 2013. We have tried to fix all occupancy demands and for two students we have introduced some hard repair constraints for some days off or a late shift. The original schedules can be found in appendix F, the repaired schedules in appendix G.1.
The first presented roster (page 139) is the repaired version of the roster on page 133. The second presented roster (page 141) is the repaired version of the roster on page 136.

### 6.8.1.1 Results of repairing occupancy in the first roster

It can be seen that most employee shortages have been resolved, except a night shift period during a weekend and an SVK shortness during a day shift on Friday March 15. For the night shift period, the algorithm was not able to allocate an MC-qualified nurse, but allocated a VK-qualified nurse instead. Hence, the total number of assigned employees satisfies the demands.

The reason for the SVK-shortness is that the number of senior nurses is low and an SVK employee requested for a quality day on this particular day. Since employees having a quality day may not be counted for the total demands, the algorithm tried to allocate 3 senior nurses instead of 2 , which unfortunately failed.

However, there is a simple manual solution by swapping the day allocation of MC employee 16 with the late allocation of SVK employee 1. Since the repair algorithm did not consider such a step, this action was not performed.

### 6.8.1.2 Results of repairing occupancy in the second roster

It can be seen that all employee shortages have been resolved, except an SVK shortness during a day shift on Friday February 15. The reason is the same as above. Again, there is a simple manual solution by swapping the day allocation of MC employee 8 or 14 with the late allocation of SVK employee 1.

### 6.8.1.3 Influences on the students' roster quality

We have added some additional days off for student 1 during the last week of the roster. In the first schedule (page 139) we could see that the workshifts are perfectly moved to the other weeks of the schedule. However, we could also see that the second week of the period remains a very quiet one for this employee.

We have therefore also tried to switch the rules consecutive workdays and current number of workdays during the week of allocation in the sorting order of the ProvideWorkDay() function, but this resulted in only a single workday during the second week of the schedule and we do not know whether this is really a more desired situation. Manually, we would have inserted at least 2 day shifts during the second week of the schedule. We could see that the sorting of the ProvideWorkDay () function, although it looks quite plausible, does not perform the best possible action for all situations.

Also the schedule of student 2, who obtained some additional late shifts, has been resolved well, but we see that a single day off is introduced here.
The same observations hold for the repaired situation (page 141) of the other schedule.
In the end, a change of the desired shift is not a very hard problem to resolve for a student, since students are largely added as surplus to the workshifts and therefore do not have to be replaced by another employee.

### 6.8.2 Additional experiment with repairing rosters

We have performed some additional experiments with roster repairing. Our original input schedule was the first presented roster without any already performed repair operations (page 133). Besides the operations mentioned in the previous subsection, we also tried to insert some other realistic constraints.

- MC employee 5 would like to change his shift allocation on the following days:
- Monday February 18 should be a day off (instead of late).
- Tuesday February 26 should be a day shift (instead of late).
- Monday March 11 should be a late shift (instead of day).
- MC employee 15 should perform day shifts on both February 19 and February 20.
- VK employee 5 would like to have a small holiday from February 26 until March 3.

The result can be found on page 144 in appendix G.2.
We could see that the repair heuristics are able to perfectly resolve all problems (except for the night shift period mentioned before). This is largely done by allocating the late shifts to the remaining flexible employees. In this way, we have prevented that other employees obtain worse schedules.

MC employee 18 obtains a block of night shifts and two days off and therefore needs an additional workshift. This employee is perfectly used to fill up the late shift shortness introduced by MC employee 15 on Monday February 18.

We also see that the allocation of days off in the schedules of MC employee 15 and VK employee 5 is done very well. A day shift on the boundary of a block of workshifts is removed and the dispersion of workshifts in the rosters remains good.

MC employee 5 obtains two single days off as a result of the ResolveLateDay() operations. We could not really prevent this for Monday February 18. However, in order to prevent the single day off at Tuesday March 12, we could maybe perform some shift exchanges with other employees and provide a late shift for MC employee 5 . Of course we could also directly provide a late shift, but then we will have the situation that the employee is surplus. Since MC employee 5 requested the late shift after scheduling the roster, we personally think it is more fair to provide an unwanted single day off instead of an unwanted overflow of employees. In this way, the bad properties of repairing schedules do only have a consequence on the schedule of the 'pollutant'.

We have also performed an additional experiment in which we did not allow the allocation of shifts to flexible employees (page 146). We could see that MC employee 9, 18, 20 and 23 obtain a schedule change in order to try to satisfy most demands. Again, a single day off is sometimes allocated in order to resolve a Late $\rightarrow$ Day transition.

Unfortunately, we also introduced a problem during the day shift on February 23 and 24 (both weekend days). Since we could not use a flexible employee for resolving a shortness during the late shifts (introduced as a result of a night shift repair), we could only use an employee who already performed a day shift during that weekend to repair this. This employee was allocated to the late shift, but the algorithm was not able to repair the day shift any longer.

### 6.8.3 Conclusion and discussion

We have seen that our repair heuristics work quite well to resolve most or all shortages, depending on the actual situation. The normal shortages could largely be resolved, flexible employees are efficiently given additional workshifts and the overall quality of the employee schedules remains rather good.

Also the allocation of additional hard repair shift requirements is performed well, but the individual schedules start to become of worse quality when the number of repair operations increases. Hence we should prevent too much repair actions.

We could take another look at resolving Late $\rightarrow$ Day transitions. Currently, undesired single days off are quickly introduced as a result of how the ResolveLateDay () repair system is build.

Also the allocation of additional workdays (ProvideWorkDay ()) is sometimes done in a busy week instead of a very quite one, since it tries to extend a current block of workshifts. However, it remains to be seen whether we should really adjust the sorting order of this recovery utility, since not allocating single workdays during a quiet week might be more preferred in some situations than allocating a single workday somewhere in the schedule.
We could also try to introduce a function which allocates (a block of) two days off at a single time. This could be used in order to prevent the introduction of single workshifts in some situations. To make this work, we have to re-examine the current situation in which a shortness in workshifts is resolved directly. Perhaps we could better restore the number of workshifts for an employee during a later stage of the repair actions.
Moreover, we could extend some of the sorting methods with additional rules (especially for the functions ProvideWorkDay () and ResolveLateDay ()) or we could try to perform different actions for different kinds of employees by introducing a set of repair preferences for every employee. The obvious drawback of this approach is again an additional parameter set, while the current number of parameters is already quite large.

In any case, the repair system remains a tool which tries to introduce as many (additional) desired properties as possible. When the tool is unable to satisfy some demands, the scheduling employee can still try to perform the desired changes manually. He has the final word about the schedules anyway.

## Chapter 7

## Conclusion and discussion

In this final chapter of this thesis report, we will draw our research conclusions and discuss the positive and negative aspects of our results. Furthermore, we will provide pointers for future research.

### 7.1 Conclusion

We will first repeat the main goal of this project, as defined in definition 1.6 on page 17 .
Definition 7.1 (Main goal). Create a technique that is able to create an employee work schedule for the nurses of the department of cardiothoracic surgery of the UMC. Both the provided department and employee preferences and requirements should be satisfied as much as possible.

We have developed a solution which consists of 5 steps:

1. Identify and insert all personal preferences for all nurses.
2. Create a set of possible rosters for each employee (except for some flexible employees).
3. Combine the created rosters into a total schedule which satisfies all occupancy demands and possibly some additional demands. An (Integer) Linear Program is used for this step.
4. Try to repair shortness in shifts using repair heuristics.
5. Try to introduce other desired properties using the same set of heuristics.

### 7.1.1 Creation of individual rosters

The creation of individual rosters could be performed using a depth-first approach in combination with a scheduling pattern and some bounding rules. The DFS-approach tries to create schedules which satisfy the properties of a forwards rotating roster pattern, while the bounding rules prevent illegal shift allocations. This could be both allocations forbidden by the management of the hospital (e.g. illegal appearances of night shift blocks, too many night shifts or too many workdays) and undesired situations for the employee (unpreferred shifts, too many consecutive workweekends, etc.). Since most shift preferences are introduced as hard constraints while generating and the worst rosters (highest costs) will be removed, the quality of the final set of generated rosters for an employee is rather high. This is the great advantage of our generation system.

In order to obtain a manageable - but large enough - number of rosters, a personal generation scheme is needed for every employee. Unfortunately tweaking the parameters of such schemes could be a hassle and sometimes a combination of realistic demands is difficult to introduce.

Besides, some employees with special demands (hardly having a forwards rotating roster pattern or having a roster which consists of small shift blocks) are difficult to roster using such a scheme. An alternative approach is needed in order to schedule them.

### 7.1.2 Combining rosters using an ILP

For the third step, an LP-relaxation is solved first using the techniques of reduced simplex (iteratively introducing the best rosters as a column in the LP, according to the reduced costs). The result is an indication for the best possible solution, although it is not a real solution. Thereafter, the final solution of the ILP is calculated using the default CPLEX-methods (e.g. branch and bound and branch and price).

Only a small set of columns (rosters) is taken into account while solving the ILP. These are all generated columns which were introduced as described above while solving the LP-relaxation. All other columns are ignored.

This set of columns can be extended in order to achieve a better ILP result. Two additional steps are therefore performed prior to calculating the ILP result:

- A set of additional rosters (columns) is added to the LP. These rosters were already generated during step 2, but not introduced since it was not needed for obtaining the best LP solution. However, the columns can be helpful to obtain a better ILP solution.
- All rosters used in the optimal LP result are mutated and all new rosters are also introduced as a column. Furthermore, a selection of the best rosters which were added as a column during the LP phase, but not used in the final result, is mutated. The best rosters, resulting from these mutations, are also introduced as a column.

The mutation step has a positive effect on the quality of the final total schedule, but we must be careful to use it. We should only introduce a selection of small mutations of already existing columns, since the effects will otherwise be very negative.

Introducing too many changes to existing rosters will usually not lead to rosters which are suitable to use in the ILP solution, given the already existing rosters. Furthermore, the number of possible offsprings increases considerably when more mutation operators are successively applied to a single roster. It takes a lot of time to create all these offsprings and to create the best selection of mutated rosters to use in the ILP solution.

We can not take too many new rosters into account, since we will then experience major solving issues while solving the ILP, because the search space for the branch and bound algorithm will be increased enormously. We have experienced that the time needed to solve the ILP will be considerably higher when we use too many mutations instead of no mutations at all.

### 7.1.3 Repair heuristics

The developed set of repair heuristics constitutes a good tool for introducing some desired properties which were not satisfied during step 3. Especially shortness in (night) shifts could be resolved. This works very well in general and the resulting rosters are still quite good (especially since we use the flexible employees efficiently), as long as the number of repair operations to be performed remains small.

However, a high number of changes (especially resolving Late $\rightarrow$ Day transitions) may lead to individual employee rosters with (much) less quality. Therefore, we should only use the repair heuristics for necessary operations and try to insert most demands in the ILP stage.

### 7.1.4 Final conclusion

We conclude that an (Integer) Linear Program is a powerful method to resolve the nurse scheduling problem with qualification constraints. It performs very well when the input (a selection of possible rosters for every employee) is varied enough to satisfy most constraints. The real challenge that remains is the problem of roster creation. Therefore, our solution could be seen as a proof of concept which has to be expanded further.

Obtaining rosters which both satisfy most employee preferences and are rich enough to use in the ILP, is difficult. A method which works in all (realistic) circumstances is hard to develop. Nevertheless, our own method works quite well for the most important kinds of demands and employee preferences, as long as the number of exceptional demands is low. Manually tweaking the parameters is needed and could be a hassle, but in the end most employee preferences and requirements of the UMC could be satisfied.

### 7.2 Discussion

Employee scheduling - and especially nurse rostering - has already been researched for years. It is a difficult subject, although a lot of solutions have been developed. As discussed in chapter 2, most solutions consider only very simple situations with a small number of constraints. In case a more realistic situation is discussed, then most of the times either:

1. A complex or expensive solution method is designed, but it only works for very small ward sizes (or the solution is so complex that it could not be used at all in practice).
2. The solution still lacks a couple of real-life constraints. Especially the employee's preferences for (no) allocation of particular shifts are only considered as 'desired', not as (quite) hard constraints.
3. The set of nurse qualifications is considered as distinct (could not overlap), hence the problem is solved in isolation for each distinct set of employees with the same qualification.

We have already mentioned the very recent paper by He and $\mathrm{Qu}(2012)[14]$, which is largely based on research in their previous publication (He and Qu (2008)[12]). Essentially, their solution which was not known to us at the time we developed our method - has a lot in common with our solution. Constraint programming (formulated as an ILP) is used in combination with column generation. Moreover, rosters are also partly created using a bounded variant of depth-first search. Still there are some major differences.

First, although the authors distinguish different categories of nurses at first sight, they do not really consider qualifications. The categories introduced in this paper[14] are largely based on the working contracts of the nurses. Hence there are only non-overlapping categories for employees, for instance those with a working contract of 36 hours per week, with 32 hours per week, and with 20 hours per week.

Second, rosters are not generated for a single employee, but for a single category as a whole. Therefore, a nurse with a contract of 24 hours per week and a preference for Thursdays and Fridays off will obtain his schedule from the same roster set as another employee with the same contract size but a preference for working on Thursday and Fridays. Shift preferences (e.g. no late shift on a particular day) are only treated as soft constraints: a penalty is given if a roster does not satisfy a constraint. Our own experience is that the employees will either be assigned to rosters in which at most a small proportion of their preferences is satisfied, or an incalculable number of schedules must be generated to deal with all requests of all employees.

Third, some really hard constraints are also considered as soft ones. Illegal shift transitions (a day shift succeeding a late shift, too few days off after a block of night shifts, incomplete weekends) are also penalized as a soft constraint. In the end, the resulting rosters may have a lot of these undesired properties.

However, the authors of the aforementioned paper conclude that a combination of constraint programming and column generation is effective and efficient, which supports our own findings that linear programming is a very powerful technique for combining schedules.

Of course our findings are also endorsed in the key paper by Hoogeveen and Penninkx (2007)[17], described in section 2.3. The nice benefit of the aforementioned paper is that the authors successfully applied the technique of roster combination using an ILP in a partly different field of employee scheduling. It is however impossible to compare the roster generation schemes and additional features (such as successive schedules) to our solution, since their rosters were designed for a very constrained work environment in which the number of employee preferences to allocate was low. Since both a simple pattern and a short roster period could also be used for their roster generation system, developing (successive) schedules was quite easy compared to our situation.

### 7.2.1 Level of difficulty

What makes nurse scheduling - and especially the situation in the UMC Utrecht - difficult?
First of all, there is a very large number of employee preferences (among the enormous number of default scheduling constraints considered in most 'realistic' nurse scheduling problems). The UMC Utrecht would like to be an excellent employer and therefore provide each employee a roster they like. Our personal feeling is that both the extent to which employees may influence their roster and the number of preferences taken into account by the scheduling employee is enormous compared to other $24 / 7$ working environments, although it was stated during the first consultation that an employee may only have 6 hard preferences among all general preferences (such as 'no single days off', which can hardly be satisfied for some employees with a lot of annoying preferences).

When all individual (shift) preferences are combined with all different sizes of the working contracts, we will see that every employee will have a kind of personal roster which can hardly be generalized into a single generation system. We have tried it and partly succeeded, but even then there are lots of possible exceptions which could hardly be taken into account.

Special generators (such as the predefined weeks generator) are introduced to overcome the generation problems for some employees. It is a nice benefit that it works, although it could only be used for very restricted employees, since we are in fact permuting all possible shifts for six weeks. In case some extension is added to the normal DFS generation system, the special generators should also be modified, which is sometimes not an easy task.

During an intermediate meeting with our client in the UMC, we were told that the results looked promising, but there were also some things they would rather see (a little bit) different, although many aspects were already taken into account and their handcrafted schedules (created in 5 workdays) also did not satisfy all demands. Furthermore, we have experienced that the scheduling employee finds it difficult to express what kinds of schedules he considers to be good.

Moreover, it is hard to express the quality of the individual rosters and the total (combined) roster into a (single) cost function, since the overall roster quality is determined by many aspects. Therefore, a good generation or repair system which is based on applying mutations is hard to develop.

For the repair phase, we have first considered outlining various situations. We tried to create a system which checked whether an employee meets a predefined shift arrangement and then performed the best possible actions accordingly. These shift arrangements were sorted from very preferred (ideal) to use to less preferred. In this way, it was tried to perform the best 'repair action' in every situation, in order to more or less guarantee that the roster quality remains high.

However, the number of possible situations and corresponding actions is such high, that it becomes impossible to enumerate them all. Only enumerating the most relevant ones also seemed to be a lot of work and it actually became clear that most situations usually did not show up in the employees' rosters, making the repair system not very valuable.

In the end we have chosen to introduce a 'simple' preference system for selecting an employee to perform a particular action. This system is only based on some essential aspects of employee rosters: no violation of very hard constraints and hopefully easy recovery of some undesired situations. Only very hard shift requirements (the 'permanent' ones) are taken into account while considering a step, all others might be ignored, although you would like to take them into account too. Other preferences such as shift block lengths and the number of occurrences of a particular shift are not considered at all.

Another aggravating circumstance in the UMC case is the large roster period of 6 weeks. For a period of 3 or 4 weeks, the number of generated rosters for each individual employee is much lower, hence we could apply a more 'relaxed' general generation scheme than in the six week situation. It is then more likely that the preferences of most employees could largely be allocated. However, as discussed in chapter 4, a shorter period was rather incompatible with the requirements which were defined for periods of six weeks.

Finally and most important: while creating a solution system, you would like to guarantee that it actually works with every realistic input and even out of the box. Although the field of optimization could not guarantee a very good solution, you definitely want to have 'a' solution in every circumstance.

While experimenting, we found that inserting an additional realistic constraint may easily drop the number of generated rosters for an employee to (almost) zero. However, every employee should obtain at least a single generated schedule in order to be able to calculate a solution. Manually tweaking parameters is therefore needed in our solution, but is in fact highly undesired. Furthermore, the effects of tweaking the parameters of an employee are sometimes quite unpredictable.

### 7.2.2 Our contribution and open problems

Although it is needed to set the parameters of all employees manually, the results look quite promising. Of course the generated schedule should be modified manually to insert some additional demands, but we think this could not be prevented in any rostering program.
Moreover, the currently used 'professional' roster platform (Monaco P.I.P.) is not used for automatic scheduling, but only for the administration of hours worked. According to an insider, the way the program schedules employees is rather stupid. Therefore, it is not strange that the scheduling employee states that he is highly unsatisfied with the rosters of the automatic scheduling tool (hence he does not use the tool). Although we did not explicitly compare the results of Monaco to ours, it is plausible that our technique performs much better, since the UMC was rather satisfied with our results.

For most employees, only a few parameters have to be changed from a default setting in our solution (the example settings in appendix F have a lot in common), but it is sometimes difficult to figure out which ones are sufficient for an employee. On the other hand, once this is done for a period, the same settings could be used for the next period, as long as the preferences stay the same and the incidental shift requirements are not conflicting to the roster pattern being used. In case some incidental shift requirements could not be inserted directly, our repair system is able to automatically resolve most issues, while generally keeping a high roster quality, as long as the number of changes is limited.
The linear programming approach is extremely strong and could take a lot of complex puzzling off your hands. Automatic individual roster generation remains an open problem, but it could become easier in the future. Our experiments were performed on a computer with 12 GB Ram (of which 8 GB was needed for the calculation) and 4 CPU cores, but expensive state of the art equipment which is also used for weather forecasting is already available. Although you do not want to use a supercomputer for a single department of 50 employees, it could be tried to setup a cloud architecture dedicated to solve employee scheduling problems in a lot of companies. Every company can select the regulations they want to have and a quite powerful computer (or perhaps an outdated supercomputer) can calculate the resulting schedule.

Also more advanced computer equipment for home and office use will become cheaper in the next years. This does not mean that we are able to generate much more rosters for a 6 weeks period, but still we could generate some more.

As already said, the possibility to generate more rosters entails the possibility to use a more relaxed generation scheme. The latter entails that the extent of manual tweaking will be reduced.
Other open problems remain: the assignment of employee collaboration, employees sharing half weekends (e.g. one employee performs the Sunday shift when the other one performs the Saturday shift) and similar situations. You would definitely take them into account while generating a schedule instead of tweaking it afterwards.
Furthermore, additional demands such as quality days (or other working tasks whereby the assigned employees do not count for the normal occupancy demands) could be inserted into the ILP, but you should remain careful since ILP's will be hard to solve when they become much more complicated. Also lots of overlapping qualification constraints could extend the number of linear program rules tremendously, as is discussed in chapter 3.
Finally, exceptions such as a switch of the appointment size or preferences halfway the period are difficult to take into account. It entails a lot of work to program all possible exceptions and it is hard to include them in a single generation scheme.

### 7.3 Future research

The final section of this thesis will provide some pointers for future research. The main question here is 'How should nurse scheduling be performed in the future?', although we will largely discuss the problem of roster generation, since a linear programming approach is a very powerful and suitable method to use.

### 7.3.1 Automatic parameter tweaking

A possible research is the investigation of the parameter sets. As can be seen in appendix F , most of the individual parameter sets are very similar. Only a few parameters have to be changed from a default setting in order to create a calculable and sufficient number of schedules. The question is: which ones?

We could investigate what kind of parameter changes are possibly appropriate for what kind of employees. Using this information, we could try to introduce an automatic parameter tweaking tool which tries to obtain approximately 200.000 generated schedules per employee. The tool tries to perform the most promising parameter changes and checks the number of generated rosters. When the number is too low or exceeds a predefined upper bound, another change must be performed (and perhaps the previous change must be undone).

### 7.3.2 Constraint programming

Another possibility is to fix all shift requirements (preferences for day and late shifts and days off on particular days) and then use a constraint programming approach to generate a set of promising schedules for every employee. This may especially work for employees who have many personal (shift) constraints.

We create a set of typical, common shift blocks of different lengths in advance. These blocks can be used to fill up the gaps between the fixed shift requirements. For each block of days for which the shift is still undefined, we fill up the gap with a shift block of enough length from the set (or a combination of blocks in case of a long period). However, this must be done according to strict rules. In any case the general hard requirements which hold for employee schedules should
be satisfied. These hard requirements can be included as constraints in a constraint programming solution.

A possible drawback of this approach is that it will be harder to check and guarantee some nice properties of the schedules (such as obtaining a 'nice' forwards rotating pattern), which was 'quite easy' in the DFS-situation.

### 7.3.3 Other suggestions

We could introduce a bounding rule which tries to create a schedule with two equal parts. While generating rosters, we only allow schedules having the last three weeks equal to the first three weeks. The high number of preferences for shifts on particular days of the week lies at the base of this idea. In case a difference of 0 allocations does not give a solution, we could iteratively allow an exception for $1,2, \ldots, 21$ days until we have obtained a sufficient number of schedules.

In this way, we could probably use a more relaxed default generation scheme, but the question that remains is how the ILP phase will respond to it. Also allocation of night shift blocks of 3 or 4 days becomes hard. Especially employees who prefer a block of 4 consecutive night shifts will obtain less preferred rosters, since there is a smaller chance a block of 4 night shifts will be allocated.

Furthermore, we could perform experiments in which the number of allocated day or late shifts must be within certain bounds (e.g. only accept schedules with at most 4 late shifts or at least 10 day shifts). But we will then introduce more and more bounding rules, making the setup more complex.

Our final note will be on repairing rosters. This is also a research area which can still be examined well. Our own introduced approach works well when the number of changes is small, but it could be examined whether additional (more complex) rules could be developed in order to keep a higher schedule quality. Perhaps a kind of roster quality measure (cost value) could be developed in order to try local search techniques. However, you should always prevent the introduction of repair cycles in which resolved problems will be re-introduced after a few steps to resolve another problem.

## Bibliography

[1] M. Bazaraa, J. Jarvis, and H. Sherali. Linear Programming and Network Flows. Wiley, New York, 1990.
[2] F. Bellanti, G. Carello, F. Della Croce, and R. Tadei. A greedy-based neighborhood search approach to a nurse rostering problem. European Journal of Operational Research, 153:28-40, 2004.
[3] E. Burke, P. De Causmaecker, and G. Vanden Berghe. A hybrid tabu search algorithm for the nurse rostering problem. In Proceedings of the Second Asia-Pacific Conference on Simulated Evolution and Learning, pages 187-194. Springer, 1998.
[4] E. Burke, P. De Causmaecker, G. Vanden Berghe, and H. Van Landeghem. The state of the art of nurse rostering. Journal of Scheduling, 7:441-499, 2004.
[5] E. Burke, P. Cowling, P. De Causmaecker, and G. Vanden Berghe. A memetic approach to the nurse rostering problem. Applied Intelligence, 15:199-214, 2001.
[6] E. Burke, T. Curtois, G. Post, R. Qu, and B. Veltman. A hybrid heuristic ordering and variable neighbourhood search for the nurse rostering problem. European Journal of Operational Research, 188:330-341, 2008.
[7] T. Cormen, C. Leiserson, R. Rivest, and C. Stein. Introduction To Algorithms. MIT Press, Cambridge, Massachusetts, third edition, 2009.
[8] K. Dowsland. Nurse scheduling with tabu search and strategic oscillation. European Journal of Operational Research, 106:393-407, 1998.
[9] A. Ernst, H. Jiang, M. Krishnamoorthy, and D. Sier. Staff scheduling and rostering: A review of applications, methods and models. European Journal of Operational Research, 153(1):3-27, 2004.
[10] M.C. Ferris and A.B. Philpott. On the performance of karmarkar's algorithm. The Journal of the Operational Research Society, 39(3):257-270, 1988.
[11] Genootschap Onze Taal. Een mens lijdt dikwijls het meest....
http://www.onzetaal.nl/taaladvies/advies/een-mens-lijdt-dikwijls-het-meest, Visited: 16-08-2012.
[12] F. He and R. Qu. A hybrid constraint programming approach for nurse rostering problems. In T. Allen, R. Ellis, and M. Petridis, editors, Applications and Innovations in Intelligent Systems XVI, Proceedings of AI-2008, the 28th SGAI International Conference on Innovative Techniques and Applications of Artificial Intelligence, pages 211 - 224, 2008.
[13] F. He and R. Qu. A constraint-directed local search approach to nurse rostering problems. In Proceedings 6th International Workshop on Local Search Techniques in Constraint Satisfaction, pages 69-80, 2009.
[14] F. He and R. Qu. A constraint programming based column generation approach to nurse rostering problems. Computers \& Operations Research, 39(12):3331-3343, 2012.
[15] J. Klein Hesselink, J. de Leede, and A. Goudswaard. Effects of the new fast forward rotating five-shift roster at a dutch steel company. Ergonomics, 53(6), 2010.
[16] H. Hoogeveen. Introduction column generation (master course material). http://www.cs.uu.nl/docs/vakken/stt/Aangepaste-Beun.pdf, Visited: 05-09-2012.
[17] H. Hoogeveen and E. Penninkx. Finding near-optimal rosters using column generation. Technical Report UU-CS-2007-002, Department of Information and Computing Sciences, Utrecht University, 2007.
[18] R. Hung. Improving productivity and quality through workforce scheduling. Industrial Management, 34:4-6, 1992.
[19] S. Irnich and G. Desaulniers. Shortest path problems with resource constraints. Les Cahiers du GERAD G-2004-11, Université de Montréal, 2004.
[20] B. Jaumard, F. Semet, and T. Vovor. A generalized linear programming model for nurse scheduling. European Journal of Operational Research, 107:1-18, 1998.
[21] N. Karmarkar. A new polynomial-time algorithm for linear programming. In Proceedings of the sixteenth annual ACM symposium on Theory of computing, STOC '84, pages 302-311, New York, NY, USA, 1984. ACM.
[22] R. Karp. Reducibility among combinatorial problems. In R. Miller and J. Thatcher, editors, Complexity of Computer Computations, pages 85-103. Plenum, New York, 1972.
[23] K. ter Laan. Letterkundig woordenboek voor Noord en Zuid. G.B. van Goor Zonen's Uitgeversmaatschappij, Den Haag / Djakarta, second edition, 1952.
[24] S. Rusell and P. Norvig. Artificial Intelligence A Modern Approach, chapter 4, pages 110-119. Pearson Education, Inc., Upper Saddle River, New Jersey, second edition, 2003.
[25] J. Tien and A. Kamiyama. On manpower scheduling algorithms. Society for Industrial and Applied Mathematics, 24(3):275-287, 1982.
[26] UWV (Uitvoeringsinstituut Werknemersverzekeringen) in collaboration with the municipal government. Arbeidsrecht - flexibele arbeidsvormen. https://www.werk.nl/werk_nl/ werknemer/meer_weten/arbeidsrecht/flexibelearbeidsvormen, Visited: 20-08-2012.
[27] M. Warner. Nurse staffing, scheduling, and reallocation in the hospital. Hospital \& Health Services Administration, pages 77-90, 1976.
[28] Wikipedia. Shadow price. https://en.wikipedia.org/wiki/Shadow_price, Visited: 16-10-2012.
[29] Wikipedia. Linear programming. https://en.wikipedia.org/wiki/Linear_programming, Visited: 30-08-2012.

## Appendix A

## Extraction of the CBA

The next pages will provide an extraction of the provided Collective Bargaining Agreement, which is used by the UMC Utrecht. It must however be noticed that many rules in the department of research are stricter than defined in the CBA.

The full document consists of 8 pages. We have only included the pages $2-5$.

|  | VEREENVOUDIGDE ATW algemeen | VEREENVOUDIGDE ATW/ATB m.b.t. verpleging en verzorging | VEREENVOUDIGDE ATW/ATB m.b.t. artsen |
| :---: | :---: | :---: | :---: |
| Mededelingstermijn van arbeids- en rusttijdenpatroon | 28 dagen $^{*}$ van tevoren. Indien i.v.m. aard arbeid niet mogelijk worden rusttijden 28 dagen van tevoren medegedeeld en 4 dagen van tevoren begin- en eindtijden van de arbeid. <br> Bij collectieve regeling óf instemming individuele medewerker kan hiervan worden afgeweken. * afwijking CAO UMC $=10$ kalenderdagen zie art. 6.3 lid 3 . |  |  |
| Minimumrusttijden: <br> -per week (5:5 ATW) <br> -dagelijks (5:3 ATW) | -36 uur per periode van $\mathbf{7 \times 2 4}$ uur of 72 uur per periode van $14 \times 24$ uur, op te splitsen in perioden van minimaal 32 uur <br> - $\mathbf{1 1}$ uur per 24 uur ( $1 \times$ per periode van $7 \times 24$ uur in te korten tot 8 uur) | (5.19:2 ATB) <br> -11 uur per 24 uur ( $1 \times$ per periode van $7 \times 24$ uur in te korten tot 8 uur alsmede tot 10 uur*) <br> *collectieve regeling |  |
| Zondagsarbeid: (5:6 ATW) <br> -arbeidsverbod <br> -vriie zondagen | -geen arbeid tenzij....... <br> tegendeel is bedongen/vloeit voort uit aard arbeid of noodzaak wegens bedrijfsomstandigheden (instemming OR/medewerker) <br> -tenminste 13 per 52 weken* <br> $\rightarrow$ *afwijkende CAO bepaling: -medewerker heeft in een kalenderjaar recht op 22 vrije weekeinden (art. 6.3.1 CAO UMC) |  |  |
| Maximumarbeidstijden:(5:7 ATW) <br> -per dienst <br> -per week <br> -4 weken <br> -per 16 weken | - 12 uur <br> - 60 uur <br> - gemiddeld 55 uur p.w.* <br> - gemiddeld 48 uur p.w.* <br> * afwijken mogelijk bij collectieve regeling |  |  |


|  | \|VEREENVOUDIGDE ATW algemeen | VEREENVOUDIGDE ATW/ATB m.b.t. verpleging en verzorging | VEREENVOUDIGDE ATW/ATB m.b.t. artsen |
| :---: | :---: | :---: | :---: |
| Regels bii nachtdienst: (5:8 ATW) | (=dienst waarin > dan 1 uur gewerkt wordt tussen 00:00 uur en 06:00 uur) |  |  |
| -rust na nachtdienst (eindigt na 02:00) -rust na reeks 3 of meer nachtdiensten | -14 uur $\rightarrow 1 \times$ per $7 \times 24$ in te korten tot 8 uur -46 uur |  |  |
| maximum arbeidstijd nachtdienst -per nachtdienst | -10 uur (max. $5 \times$ per $14 \times 24$ en max. 22 per 52 weken te verlengen tot 12 uur (rust na verlengde nachtdienst min. 12 uur)) |  |  |
| -per week | -60 uur |  |  |
| -per 4 weken | -gemiddeld 55 uur p.w.* |  |  |
| -per 16 weken | -bij >dan 16 nachtdiensten gemiddeld 40 uur p.w.* |  |  |
| maximum aantal nachtdiensten: -per 16 weken | $-m a x$. uur 36 nachtdiensten*: w.v. einde na 02:00 |  |  |
| -per 52 weken | -max. 140 nachtdiensten w.v. einde na 02.00 uur of 2 aaneengesloten weken max. 38 uur arbeid vervult tussen 00.00 en 06.00 uur) |  |  |
| -max. aantal per reeks nachtdiensten | -geen norm |  |  |
| -max. aantal achtereenvolgende diensten in reeks met 1 of meer nachtdiensten | $\begin{array}{\|l\|} \hline-7 \\ \text { *afwijken mogelijk bij collectieve regeling } \end{array}$ |  |  |
| Pauze:(5:4 ATW) <br> -arbeidstijd per dienst > $51 / 2$ uur | -minimaal $1 / 2$ uur (of $2 \times 1 / 4$ uur) |  |  |
| -arbeidstijd per dienst > 10 uur | -minimaal $3 / 4$ uur (evt. in pauzes van $1 / 4$ uur) |  |  |



|  | VEREENVOUDIGDE ATW algemeen | VEREENVOUDIGDE ATW/ATB m.b.t. verpleging en verzorging | VEREENVOUDIGDE ATW/ATB m.b.t. artsen |
| :---: | :---: | :---: | :---: |
| speciale consignatie in vorm van Bereikbaarheidsdienst: -maximaal aantal diensten |  | (5.19:3 ATB) -max. 3 per periode van $7 \times 24$ uur en 32 per periode van 16 weken | (5.20:4 ATB) <br> -max. 5 per periode van $7 \times 24$ uur en 32 per periode van 16 weken |
| speciale consignatie in vorm van Aanwezigheidsdienst: -maximaal aantal diensten <br> -minimum onafgebroken rust voor en na aanwezigheidsdienst |  | (4.8:1 ATB) -max. 52 per periode van 26 weken <br> (4.8:1 ATB) <br> - $\mathbf{1 1}$ uur ( $1 \times$ per periode van $7 \times 24$ uur in te korten tot 10 uur alsmede 1 x tot 8 uur ) | (4.8:1 ATB) <br> -max. 52 per periode van 26 weken <br> (4.8:1 ATB) <br> -11 uur (1x per periode van $7 \times 24$ uur in te korten tot 10 uur alsmede 1 x tot 8 uur ) |
| -maximale arbeidstijd per periode van 26 weken <br> -cumulatie bijzondere diensten <br> (consignatie + aanwezigheid en/of bereikbaarheid) |  | (4.8:1 ATB) gemiddeld 48 uur p.w. | (4.8:1 ATB) gemiddeld 48 uur p.w. |
| maximum aantal diensten : <br> -per 7x24 uur <br> -per 16 weken |  | $\begin{aligned} & \text { (5.19:4 ATB) } \\ & -\mathbf{3} \\ & \mathbf{- 3 2} \end{aligned}$ | $\begin{aligned} & \text { (5.20:5 ATB) } \\ & -5 \\ & -\mathbf{3 2} \end{aligned}$ |

## Appendix B

## List of duty codes

These are the most important codes used in the rosters of the UMC. The word 'Day' refers to the early shift. The Dutch translation is sometimes added between brackets for clearness reasons.

Students (STD)

| DF | Day shift Student |
| :---: | :--- |
| A4 | Late shift Student |

Basic Nurse (VPK)

| D8 | Day shift Basic Nurse |
| :--- | :--- |
| DE | Day shift Trainee Nurse |
| A8 | Late shift Basic Nurse |
| N8 | Night shift Basic Nurse |

Medium Care Nurse (MC)

| D1 | Day shift MC Nurse |
| :--- | :--- |
| DG | Day shift MC Nurse In Training |
| A7 | Late shift MC Nurse |
| N1 | Night shift MC Nurse |

Senior Nurse (SVK)

| DS | Day shift Senior Nurse |
| :--- | :--- |
| DK | Day shift for Quality Day |

Days off, holidays and other leave

| RV | A day off [Roostervrij ] |
| :--- | :--- |
| V8 | A vacation day [ Vakantiedag ] |
| BV | Special Leave [ Buitengewoon verlof ] |
| G8 | Maternity Leave [ Zwangerschapsverlof] |
| OO | Parental Leave [ Ouderschapsverlof ] |

Other codes

| D9 | Day shift Senior Unit Head |
| :--- | :--- |
| DC | Day shift for preparing a Student or Nurse [ Inwerken ] |
| DD | Day shift Carer [ Ziekenverzorgende ] |
| DH | Day shift Department Student [ Afdelingsstudent ] |
| Z8 | An illness day [ Ziektedag ] |
| S8 | Full day of classes [ Hele lesdag ] |

## Appendix C

## Employee preferences \& demands

This appendix shows the list of employee preferences (in Dutch) for the example roster period February 11 - March 24. It is retrieved from the scheduling employee, but changed by us thereafter since some demands appeared to be missing. It can be found on page 111.
The employees are grouped according to their highest qualification. The legend of the columns is the following:

1. The employee's name.
2. The appointment size ( 0 when flexible).
3. The employee's preferences.
4. The average deployment per week when the employee has a flexible contract.
5. Whether a special roster pattern is needed for the employee.

PR: This employee has a predefined roster (see the anonymized scheme on page 112).
PW: This employee has a roster based on predefined weeks (see the scheme on page 113).
AR: The employee may only have a schedule with 7 night shifts during in the first week and the next week off.

The table in the bottom right of the list denotes the number of shifts an employee with a certain appointment size normally performs per week. The column RV gives the number of days off the employee will retrieve during 6 weeks. If the employee's appointment size is not mentioned here, the RV value is an in between one, according to: $R V=\left\lfloor 42-\frac{3}{4} x\right\rfloor$ where $x$ is the appointment size.

Most of the possible preferences are self-evident. A few special formulations are clarified here:

- When the word 'max' is used, together with a specific shift, it means that the employee would like to have the shift for at most $x$ consecutive days.
- 'Oplopend' and 'afwisselend' refer to a forwards rotation pattern.
- 'i.o.m.' means the shift is possible in consultation.

It should however be mentioned that some 'additional' requests are not shown in the list on page 111. An impression of the incidental hard requirements (a holiday, weekend off or obliged day shift) and the holidays can be found in the anonimized scheme on page 112. Furthermore, some employees have a special informal guideline for the number of allocated late shifts. Since this information was available in a late stadium, we did not really use it.



| Name | App | Work | Off | Weekends | Shifts: |
| :---: | :---: | :---: | :---: | :---: | :---: |
| MC Employee 2 | 20 | 15 | 27 | 2 | Do laat of vrij <br> Vr laat of nacht <br> In werkweekend dag, laat (nacht eigenlijk nooit) <br> Weken (zonder nachtmogelijkheid): $\begin{aligned} & 3 x \text { do }+v r \\ & 1 x \text { do }+v r+(z a \text { of } z o) \\ & 2 x v r+z a+z o \end{aligned}$ |
| MC Employee 7 | 24 | 18 | 24 | 2 | Ma dag of avond <br> Di dag of laat <br> Wo dag of avond <br> Do vrij <br> Vr vrij (of nacht) <br> In werkweekend dag, laat of nacht <br> Weken (zonder nachtmogelijkheid): $\begin{aligned} & 2 x m a+d i \\ & 2 x m a+d i+w o \\ & 2 x m a+d i+z a+z o \end{aligned}$ |
| MC Employee 12 | 24 | 18 | 24 | 3 | Ma dag of avond <br> Di vrij <br> Wo vrij <br> Do avond <br> Vr vrij of nacht <br> In werkweekend dag, laat of nacht <br> 3 weekenden (want: om het weekend) <br> Weken (zonder nachtmogelijkheid): $\begin{aligned} & 3 x \mathrm{ma}+\mathrm{do} \\ & 3 \mathrm{xma}+\mathrm{do}+\mathrm{za}+\mathrm{zo} \end{aligned}$ |
| MC Employee 23 | 24 | 18 | 24 | 3 | Ma vrij (of nacht) <br> Di dag of avond (of nacht) <br> Wo dag of avond (of vrij bij nacht op ma + di) <br> Do vrij <br> Vr vrij <br> In werkweekend dag of laat <br> 3 weekenden (want: om het weekend) <br> Weken (zonder nachtmogelijkheid): $\begin{aligned} & 3 x d i+w o \\ & 3 x d i+w o+z a+z o \end{aligned}$ |

## Appendix D

## Example roster

An example roster is provided on the next page. The schedule dates from spring 2012, when the ward size was slightly too low.
The underlying employee soft preferences are about the same as the list provided in appendix C. The employee's own hard preferences are unfortunately not exactly known, but when an exclamation mark ('!') is used, it means the shift type was forced.

It should also be noted that the roster does not fulfill all the specified hard constrains (e.g. working a whole weekend or not). These deviations are typically final changes made by the scheduling employee for special reasons. The given roster is in fact the result at the end of the six weeks period: changes to the initial schedule have been made due to illness (' Z 8 ') and other (undefined) reasons.

Although the roster has these little irregularities, it gives a good indication of a typical unit roster.

## Legend

These are the most important legend details:

- An overview of the (most) used duty codes is provided in appendix B.
- The column PC defines the position of the employee (unimportant ones are not mentioned): $15=\mathrm{SVK}, 20=\mathrm{MC}, 25=\mathrm{MCIO}, 30 / 35=\mathrm{VPK}, \geq 60=\mathrm{STD}$.
- The column $\mathrm{U} / \mathrm{Wk}$ defines the fixed appointment size of the employee in the period. If this is 0 , the employee has a flexible contract.
- The right column and bottom rows are accusations for control reasons and are not important here.


| 19 April 2012 |  | $\begin{aligned} & \hline \text { 09/2012 [02] } \\ & 27 / 02-04 / 03 \end{aligned}$ | $\begin{aligned} & \hline \text { 10/2012 [03] } \\ & 05 / 03-11 / 03 \end{aligned}$ | $\begin{aligned} & \hline 11 / 2012[03] \\ & 12 / 03-18 / 03 \end{aligned}$ | $\begin{aligned} & 12 / 2012[03] \\ & 19 / 003-25 / 03 \end{aligned}$ | $\begin{aligned} & \hline \text { 13/2012 [03] } \\ & 26 / 03-01 / 04 \end{aligned}$ | $\begin{aligned} & \hline 14 / 2012[04] \\ & 02 / 04-08 / 04 \end{aligned}$ | $\begin{aligned} & \hline 15 / 2012[04] \\ & 09 / 04-15 / 04 \end{aligned}$ | $\begin{aligned} & \hline 16 / 2012[04] \\ & 16 / 04-22 / 04 \end{aligned}$ | $\begin{aligned} & 16 / 07 / 12 \\ & 26 / 08 / 12 \end{aligned}$ |
| :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: |
| PC. Naam | U/Wk | Ma DiWodovi Za Zo | Ma Di Wodo $\mathrm{V}_{1} \mathrm{Za} \mathrm{Z}_{0}$ | MadiWo Dovi Za Zo | Madi Wodovi $\mathrm{Za} \mathrm{Z}_{0}$ | Madi Wo Dovi Za Zo | Madi Wodovi Za Zo | Di Wo Dovi Za Zo | $\mathrm{Di}^{\text {Wo }}$ Do $\mathrm{V}_{1} \mathrm{Za} \mathrm{Z}_{0}$ |  |
|  | 34:00 | RVRV D8 D8 A8 A8 A8 | 188 HB RV RV D8 88 D8 | RV S8 S8 S8 D8 RVRV | S8 S8 A8RV D8 D8 D8 | A8 A8RVRV D8 D8 D8 | D8 A8 488 RV RV RV RV | RV 48 N8 88 RV RVRV | RVV RV D8 S88 S8 D8 D8 | 16 |
|  | 32:00 | RV RV A8888 RV D8 D8 | D8 A8 A8 RVVV D8 A8 | [18 N8 H8RV RV A888 | A8RV S8 S8 S8 RVIRV: | RV:RV:V3HVEIVE!RVIRV: |  | A | RV D8 D8 D8 RV RVIRV: | 18 |
|  | 36:00 | RV:DK D8 D8 D8 RV RV | RV:D8 D8 RV A8 D8 D8 | RV:A3PA8 ABRV RV!RV: | S8: S8 S8 RV D8 A88 A8 | RV:D8 D8 D8 D8 RV:RV: | D8! S8 D8 RV D8! D8 D8 | D8! 88 RVRV D8 RVA日 | प8! M8 48 H3RV RVIRV: | 15 |
|  | 20:00 | RV:Z8R FV : D8 RV RV RV | D8 RV D8! RV RV RV D8 | RV:D8 RV!D8!RV D8 RV | DM:ABIRV!RV RV DMD5: | D8!RV!RV!RV RV! RV RV | D8!D8!RV!RV RV D8 D8 | RVRVRV:D8!RV RVRV | RV!A8!RV!RV RV RV D8 | 27 |
|  | 36:00 | A8 V8 188 HB RV RVRV | RV DK D8 D8 RV RVA8 | A8RV D8 D8 D8 A8 A8 | RV RV D8 D8 RV:RV:RV: | W8 48888 NB RV RVRV | D8 D8 D8 RVA8! A8! ${ }^{\text {a }}$ : | A8! FR RV D8 D8! A8A8 | A3RVRV!D8 D8 RV RV | $15 \quad 13$ |
|  | 00:00 | .. D8!D8! .. | .. D8:D8! .. | .. D8:D8! RVRV | .. D8:D8: | .. D8 .. D8:D8! RVRV | .. 28 .. ... 28 R RV RV | RV .. .. .. .. D8!D8: | .. .. .. ..! ... RV RV | 19 |
|  | 32:00 | RV D8 D8 RV RV D8 D8 | A8 A8 78 NBRV RVRV | D8 D8 RVRV的 A8 AB | W8 H8RV RV D8 RV RV | D8 D8 AB7RV RV A8 A8 | RV AB A8ABRV RV RV |  | RV RV D8 D8 $A 8$ RV RV | $18 \quad 12$ |
|  | 36:00 | D8!RV!RV A8 D8 D8 D8 | D4!S8!RV!D8 D8! RV RV | D8:DCRV:A8 [18 ${ }^{\text {N8 }}$ N8 | RV:RV D8 D8 D8 RV RV | D8:DC D8 D8 RV D8 D8 | S8: S8 S8 D8 RV RVIRV: | RVIRV D8 D8 D8 RV D8 | RV:D8 D8 D8 D8! RV RV | $15 \quad 14$ |
|  | 32:00 | D8 ABTRV RV A8] A8 AB | RV S8 D8 RVRV D8 D8 | D8 A8RV D8 D8 RVRV | D8 D8 RV A88 A8 RV RV |  | RV S8 S8 D8 FR D8 D8 | D8 RV RV D8 D8 RVRV | D8 D8 RV RV D8 4888 | $18 \quad 12$ |
|  | 32:00 | RV RV D8 ABRV RV:RV: | RV:V83V8!V8HV8!RV!RV | D8 D8 A8RV RV D8 D8 |  | RVRV D8 A8 A88 RVRV | RV D8 D8 RVRV D8 D8 | D8 A8A8lRV RV RVRV | RV D8 A8 A8 A8] RV RV | 18 |
|  | 36:00 |  | RV RV S4 D8 A8B RV RV | RV D3 D8 D8 AB RVRV | D8 A8 188 N8RV RV RV | D8 A8888 RV D8 $\mathrm{D} 8 \mathrm{W8}$ | W8 NBRV RVRV RVRV | RV D8 D8 RV A88 D8 D8 |  | $15 \quad 13$ |
|  | 34:00 | D8 D8 A8 ABRV RV RV | RV S8 D8 A88 188 [88 78 | RV RV D8 D8 A8 RV RV | D8 D8 D8 RV/48] 48 D8 | RVRV D8 D8 D88 D8 D8 | RV RV RV [18RV RV RV | FRRV D8 AB RV RVRV | VBIVBIVB!V8IRV!RV!RV: | 16 |
|  | 17:17 | D8 RV RV:RV RV D8 D8 | RV RVRV:D8 D8 RV RV | D8 RVVV!RV RV D8 D8 | RV RVRV:D8 RV RVRV | D8 RV RV!RV RV D8 D8 | RV RVRVIRVRV D8 RV | D8 RVRV!D8 RV RVD8 | RV RVRV!RVRV D8 D8 | 29 |
|  | 32:00 | .. .. .. .. .. .. .. | .. .. .. .. .. .. .. | .. 58 | S8 | .. S8 .. .. | .. s8 .. .. | . | .. .. .. .. .. .. .. | 0 |
|  | 36:00 | DF S8: DF RV RV DF DF | DF S8: DF RVRV DF DF | DF S8: DF DF RV DF DF | DF S8: DF DF DF RV:RV! | S8 S8: DF RV RV 14 A4 | RV S8: DF A4 A4 RV RV | RV S8: DF RV RV DF DF | DF S8:RV RV DF RV DF | $15 \quad 14$ |
|  | 32:00 | DF S8 RV Z8888 DFA4, | RV S8: DF RVRV RVRV | RV:S8:/A4RV RV RVRV | DF S8! RV RV DF! DF A4 | S8: S8: DF DF A4 A4 A4 | A4S8:RV!RV RV DF DF | DF S8:A4RV RV! RV:RV: | A4! S8: DF DF A4 RV RV | 18 |
|  | 32:00 | A4 S8 RVRV RV RVRV | RV S8 RV AARV DF DF | A4S8: DF RV RV RV:RV: | RV S8: DF DF A4 RV RV | S8 S8!RV DF RV DF DF | DF A4:A4RVRV DF DF | DF S8:RV S8 RV A4 A4 | RV S8: A4RV RV DF RV | $18 \quad 15$ |
|  | 32:00 | RV DF DF DF DF RVRV | S8: DF DF RVRV DF DF | DF DF: DF RV RV DF DF | DF RV:DF DF DF RV RV | S8:DF:A4 DF RV RV DF | DF DF:RV DF DF RV RV | RV V8 V8 V8! ${ }^{\text {d }}$ RV RV | S8:A4RVRV!DF DF RV | 18 |
|  | 32:00 | DF RV RV S8 DF RVRV | DF DF DF S8 DF RV RV | RV DF RV S8 A4 RVRV | RV DF DF S8 RV DF DF | RV RV DF S8 DF DF DF | RV DF DF S8 RV RV RV | RV DF DF S8 DF DF DF | RV RV DF! S8 RV! RV:RV: | $17 \quad 12$ |
|  | 3200 | RV DF S8 RV RV DF DF | A4 DF S8 RVRV RV RV | RVRV S8 DF DF DF DF | RV DF S8 A4 RV RV RV | D1 DF S8 RV RV D1 Z8 | RV RV S8 RV DF A4A4! | DF!RV S8 DF DF RVRV | RV DF S8 RV DF RV RV | 18 |
|  | 32:00 | RV D8 S8 DF RV D8 D8 | D8 RVS8 D8 RV RVRV | RVRV S8 DF DF RVRV | DF DF S8 RVRV DF DF | RVRV DF DF DF DF DF | RV DF DF DF FR RV RV | FR DF S8 DF RV DF DF | RV DF S8 DFRV RV DF | $18 \quad 12$ |
|  | 32:00 | RV DF DF S8 RV DF DF | RV RVRV S8 DF RVRV | DF RV RV S8 DF DF A4 | A4RVRV S8 RV Z8 DF | DF A4RV S8 DF DFRV | DF DF RV S8 RV RV RV | RV A4 RV S8 V8] RV:RV: | V8!RV!RV S8 DF DF DF | 18 |
|  | 22:00 | D5 D5 RV D5 D5 RV RV | D5 D5 D4 D5 D5 RV RV | D5 D5 RV D5 D5 RVRV | D5 D5 D4 D5 D5 RV RV | Z5 25 RVZ ZS Z5 RVRV | Z5 Z5 Z4 d5 D5 RV RV | FR VGRV D5 D5 RVRV | D5 D5 D4 D5 D5 RV RV | 15 |
|  | 32:00 | DH DH DH DHRV RVRV | D4 DH DH DHRV RV RV | DH DH DH DHRV RVRV | DH D6 DH DHRV RVRV | DH DH DH DHRV RVRV | Z8 288888 RV RV RV | FR D4 D4. D4 RV RVRV | DH DH DH DH RV RV RV | 18 |
|  | 32:00 | .. .. .. .. .. .. .. | .. .. .. .. .. .. .. | .. .. .. .. .. .. .. | .. .. .. .. .. .. .. | .. .. .. .. .. .. .. | .. .. .. .. .. .. .. | .. .. .. .. .. .. .. | .. .. .. .. .. .. .. | 00 |
|  | 32:00 | .. .. .. .. .. .. .. | . | .. | .. .. .. .. .. | .. .. .. .. .. .. | . .. .. .. .. .. | . | .. .. .. .. .. .. .. | 00 |
|  | 32:00 | RVRV S8 DF A4 RVRV | RVRV S8 DF DF DF DF | D8 D8 RV/A8 A8 RVRV | A8 A8 A8RVRV D8RV | D8RV S8 .. D8 D8 D8 | RV D8 D8 A8, A8RVRV | FR D8 S8RVRV D8 D8 | RV D8 D8 D8 D8 RVRV | 20 |
|  | 32:00 | .. .. | .. .. .. .. .. .. .. | .. .. .. .. .. .. .. | .. .. .. .. .. .. .. | .. .. .. | . | .. .. .. .. .. .. | .. .. .. .. .. .. .. | $0 \quad 0$ |
|  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |
| Geplande Weekuren. |  | 40:00 | 40:00 | 32:00 | 48:00 | 40:00 | 24:00 | 24:00 | 40:00 | B4/02 |
| DS Dagd senior/oudste D1 Dagdienst D8 Dag-8 |  | $\begin{array}{lllllll}0 & 0 & 2 & 1 & 0 & 2 & 2 \\ 7 & 6 & 5 & 3 & 7 & 5 & 4 \\ 4 & 3 & 4 & 5 & 3 & 5 & 5\end{array}$ | $\begin{array}{lllllll}2 & 0 & 2 & 0 & 2 & 0 & 1 \\ 7 & 7 & 3 & 8 & 4 & 5 & 5 \\ 3 & 1 & 5 & 6 & 4 & 4 & 4\end{array}$ | $\begin{array}{lllllll}0 & 0 & 2 & 0 & 1 & 1 & 1 \\ 5 & 7 & 6 & 4 & 4 & 4 & 5 \\ 5 & 4 & 3 & 6 & 4 & 3 & 3\end{array}$ | $\begin{array}{lllllll}0 & 1 & 1 & 2 & 1 & 3 & 3 \\ 3 & 8 & 5 & 4 & 4 & 4 & 3 \\ 3 & 2 & 3 & 4 & 4 & 2 & 3\end{array}$ | $\begin{array}{lllllll}0 & 3 & 0 & 0 & 0 & 1 & 0 \\ 6 & 5 & 5 & 6 & 4 & 5 & 5 \\ 5 & 3 & 5 & 4 & 5 & 5 & 4\end{array}$ | $\begin{array}{lllllll}1 & 0 & 1 & 1 & 1 & 1 & 1 \\ 5 & 5 & 4 & 5 & 8 & 5 & 5 \\ 5 & 4 & 3 & 2 & 1 & 5 & 4\end{array}$ | $\begin{array}{lllllll}1 & 2 & 0 & 0 & 1 & 1 & 1 \\ 4 & 5 & 7 & 5 & 5 & 5 & 5 \\ 4 & 2 & 3 & 5 & 4 & 3 & 5\end{array}$ | $\begin{array}{lllllll}1 & 0 & 0 & 0 & 0 & 0 & 1 \\ 9 & 6 & 5 & 5 & 7 & 6 & 6 \\ 1 & 4 & 4 & 5 & 3 & 2 & 3\end{array}$ | dienst: 08:00 |

## Appendix E

## Experiments on mutation

This appendix shows the results of the experiments on mutating rosters. A few variances of mutation mechanisms were tried and compared to each other and to a baseline measurement. For some of the better results, we also tried the same setup with an extended time for solving the ILP. This was done to examine the results on the long term.

## E. 1 Setup of the experiments

We performed the experiments on the set of employees for September 2012. All employees were expected to be available. All shift preferences (the preferred shift on a particular day) were added as hard constraints, but we did not include holidays, training days and incidental shift requirements (since this information was not available for the provided period). Therefore, all shifts could easily be allocated and the optimum of the LP is low. This is quite different from the situation in appendix F. Nevertheless, appendix F provides some details on the cost settings being used.

We solved the problem in the following way:

1. Solving the LP (20 minutes allowed, less needed).
2. Intermediate phase.
3. Solving the ILP ( 15 minutes or 60 minutes allowed, needed completely).

## E.1.1 Solving the LP

The LP was solved by selecting the 100 best (lowest roster costs) columns per employee as a start. Each round, the 100 best rosters (according tot the reduced costs) for each employee were added, until no roster (with negative reduced costs) could be added anymore. The allowed solving time was 20 minutes, but it was never needed completely.

## E.1.2 Intermediate phase

The intermediate phase was divided in three stages. Stages 2 and 3 were not used in run 6 (the baseline measurement).

1. Another 2500 rosters were added to the problem instance. These rosters were already generated, but not introduced in the LP phase. The 2500 rosters were the ones with the lowest (positive) reduced cost.
2. Select all rosters which were used in the optimal LP solution. All of the rosters were mutated once or at most twice (so when the value is 2 , we both created rosters using one mutation operator and using two mutation operators after each other).
3. From the set of rosters which were added to the problem instance, the best 2500 which were not used in the optimal LP solution were mutated. Mutation was performed in the same way as stage 2. The best 2500 newly created rosters were also added to the problem instance.

## E.1.3 Solving the ILP

The ILP was solved using the default branching methods of CPLEX. The allowed solving time was varied between 15 and 60 minutes and was always needed completely.

## E. 2 Overview of the results

In the table below, the results of the experiments on mutations are shown. The optimum of the LP was 2721.95.

- The column 'Time ILP' provides the time in minutes that was allowed for solving the ILP.
- The column 'Gap' shows the integrality gap after solving the ILP.
- The column 'Mutations' shows:
- The allowed number of mutations.
- The number of new rosters that has been created while mutating the rosters which were selected in the LP result. All of them were added to the problem instance, except for run 4 and 5 in which only the best 5000 rosters have been added.
- The number of new rosters that has been created while mutating the best 2500 rosters which were added as a column in the problem instance, but not selected in the LP result. The best 2500 mutations were added to the problem instance thereafter.
- The column 'Fall' shows at which node there was a remarkable drop in the integrality gap.
- The last column shows the total number of additional columns which were added before solving the ILP. The value is equal to 2500 (extra selected columns) + the (remaining) number of mutated selected columns + the remaining number of mutated unselected columns (2500, except the last record).

| Id | Time | Mutations <br> ILP |  |  | $\#$ | Selected | Unsel.* |  | Gap <br> ILP |  | Node |  | Extent | Extra <br> col. ILP |
| :---: | :---: | :---: | ---: | ---: | :---: | :---: | :---: | ---: | ---: | :---: | :---: | :---: | :---: | :---: |
| 1 | 15 | 1 | 3,788 | 36,538 | $39 \%$ | 4445.75 | 437 | $94 \% \rightarrow 44 \%$ | 8,788 |  |  |  |  |  |
| 2 | 60 | 1 | 3,788 | 36,538 | $18 \%$ | 3301.51 | 437 | $94 \% \rightarrow 44 \%$ | 8,788 |  |  |  |  |  |
| 3 | 60 | 2 | 45,852 | 418,562 | $33 \%$ | 4049.23 | 381 | $85 \% \rightarrow 48 \%$ | 50,852 |  |  |  |  |  |
| 4 | 15 | 2 | $* * 5,000$ | 449,721 | $58 \%$ | 6470.19 | 559 | $92 \% \rightarrow 59 \%$ | 10,000 |  |  |  |  |  |
| 5 | 60 | 2 | $* * 5,000$ | 449,721 | $36 \%$ | 4241.22 | 559 | $92 \% \rightarrow 59 \%$ | 10,000 |  |  |  |  |  |
| 6 | 60 | 0 | 0 | 0 | $46 \%$ | 5005.43 | 607 | $89 \% \rightarrow 61 \%$ | 2,500 |  |  |  |  |  |

[^24]
## E. 3 Additional test on roster period February - March 2013

We also tested the behaviour of mutations on the roster period February - March 2013 (appendix F). For this period, the number of available employees was slightly less and we had more infor-
mation about holidays, teaching days and incidental shift requirements. We compared the best setting from the previous section with no mutations at all, but the allowed time for solving the LP was 15 minutes.

The results are shown in the table below. The column 'LP value before' denotes the optimal value of the LP before adding any mutations, the column 'after' denotes the improved LP solution with mutated columns. The column 'Best bound' contains the best objective function value achievable at the time the ILP solving process was cut off due to a time limit. The column 'ILP value' denotes the value of the best integral solution at the time the ILP solving process was cut off. The gap is the difference between the latter two columns.

| Mutation | LP value |  | Best bound | ILP value | Gap |
| :---: | ---: | ---: | ---: | ---: | :---: |
|  | before | after |  |  |  |
| Yes | $72,996.31$ | $69,230.09$ | $77,363.59$ | $80,992.20$ | $4.48 \%$ |
| No | $72,996.31$ | - | $88,828.95$ | $90,975.91$ | $2.36 \%$ |

## Appendix F

## Results of the ILP phase

This appendix will show two results of the ILP solution for the period February 11th until March 24th. The rosters have been created, based on the information in appendix C.
Since most flexible employees will normally be assigned anyway (one has already created her roster in advance) and a number of them have some very strict preferences (e.g. only late shifts during weekdays), some of them have already been scheduled as if they were a normal employee. This holds especially for a few of them with a large appointment size (see the column with label ' 0 ?': this is the average number of weekly hours the employee normally works each week). Others may obtain some remaining shifts during the repair phase or during manual tweaking afterwards.
The first part of this appendix will describe the input data and some parameter settings. Thereafter, we present some details about the solving process, together with the solution. Thereafter, we present some details about the solving process, together with the solution.

## F. 1 Input data

The input data for this roster is based on the lists in appendix $C$ and the occupancy demands as defined in section 1.6. We tried to insert as much information as possible, but as already mentioned in the thesis itself: finding a good parameter setting could be a hassle and sometimes it is difficult to take all demands (completely) into account.
We will not provide the full list of input parameters, since this list will be too large. However, we will provide some of the generation parameters for a selection of employees to provide an overview.

It should be noted however, that the settings for most employees are more or less the same. In most cases, only small changes in the shift block restrictions table are needed which lead to a substantial pruning of rosters for the employee. Sometimes, we need some additional changes, but this is a matter of trial and error.

Furthermore, we should mention that not all settings are relevant for the employees below (e.g. a preference for at most 4 night shifts does not mean anything when the employee is not allowed to perform night shifts, but it was needed to give the parameter a value).

- The table shift preferences shows the preferred minimum and maximum number of shifts in the roster. Furthermore, it shows the preferred minimum and maximum length of a shift block of the particular shift. This information is used as soft constraints for generating and prepruning rosters.
- The table general parameters shows some hard constraints for generating rosters. The 'extra workdays per week' are defined as the number of workdays above the (rounded up) value of the appointment size of the employee.
- The table shift block restrictions shows the shift block rules, according to which the schedules for the employee are generated. Normally, after performing a block of work shift $x$, the employee must perform a block of workshift $y \neq x$. There may be a period of days off between the two blocks (see the next bullet).
For some employees, it is necessary to have some days off inside a large block of workshifts of type $x$. Hence, they should have the shift block pattern $x \rightarrow$ Free $\rightarrow x$.

Normally this is not possible, but when the parameter 'allowed block repetition' obtains a positive value, this may occur a few times. The value of this parameter determines how many times a block of the same shifts may be performed before a switch is required. ${ }^{1}$

- The table shift pattern allowance shows in which order the work shift blocks may occur (a transition to a block of days off is always allowed and therefore not in the list).
- When a pattern is not in the list, it may never occur (not directly and not with a couple of days off between them).
- When a pattern has the value always, both the pattern $x \rightarrow y$ and $x \rightarrow$ Free $\rightarrow y$ are allowed.
- When a pattern has the value day off, then only the pattern $x \rightarrow$ Free $\rightarrow y$ is allowed (at least one day off between the shift blocks).
- The final tables show the soft and hard constraints for the period. The value 'Counts demand' may only be set to 'no' in case the type is 'always' and the value of 'assign' is yes. It is used to provide a day shift to employees who have a study day and hence are not available for nursing tasks. In this way, the day shift fits into the pattern of the employee, but he will not be counted for the occupancy demands.

[^25]
## F.1.1 SVK employee 1 (SVK, $32 \mathrm{hr} / \mathrm{wk}$ )

Remark: Since SVK employees are needed for the day shifts, we decided to change the demand of having 4 night shifts in having 2 night shifts.

## F.1.1.1 Shift preferences

| Shift | Min count | Max count | Min length | Max length |
| :--- | ---: | ---: | ---: | ---: |
| Day | 0 | $\infty$ | 0 | $\infty$ |
| Late | 0 | $\infty$ | 0 | $\infty$ |
| Night | 0 | 2 | 2 | 2 |
| Free | 0 | $\infty$ | 0 | $\infty$ |

## F.1.1.2 Generation model

## General parameters

| Property | Value |
| :--- | ---: |
| Max workdays after each other: | 5 |
| Allowed extra workdays per week: | 1 |
| Max workweekends after each other | 2 |
| Single workday allowed: | No |
| Min length free block: | 2 |

## Shift block restrictions

| Shift | Block length |  | Days off |  |  |  | Allowed block repetition |
| :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: |
|  |  |  | Before |  | After |  |  |
|  | Min | Max | Min | Max | Min | Max |  |
| Day | 2 | 4 | 0 | $\infty$ | 0 | $\infty$ | 0 |
| Late | 1 | 3 | 0 | $\infty$ | 0 | $\infty$ | 0 |
| Night | 2 | 2 | 0 | $\infty$ | 2 | $\infty$ | 0 |

Shift pattern allowance

| From |  | To | Always | Day off |
| :--- | :--- | :--- | :---: | :---: |
| Day | $\Rightarrow$ | Late | $\checkmark$ |  |
| Late | $\Rightarrow$ | Day |  | $\checkmark$ |
| Late | $\Rightarrow$ | Night | $\checkmark$ |  |
| Night | $\Rightarrow$ | Day |  | $\checkmark$ |

## F.1.1.3 Soft and hard constraints

## Weekly

| Day | Shift | Assign | Weeks | Counts demand | Type |
| :--- | :--- | :--- | :--- | :--- | :--- |
| Tuesday | Late | No | All | Yes | Generation |
| Wednesday | Late | Yes | All | Yes | Soft |

Occasionally

| Date | Shift | Assign | Counts demand | Type |
| :--- | :--- | :--- | :--- | :--- |
| $11-02-2013$ | Free | Yes | Yes | Always |
| $12-02-2013$ | Free | Yes | Yes | Always |

## F.1.2 SVK employee 2 (SVK, $24 \mathrm{hr} / \mathrm{wk}$ )

## F.1.2.1 Shift preferences

| Shift | Min count | Max count | Min length | Max length |
| :--- | ---: | ---: | ---: | ---: |
| Day | 0 | $\infty$ | 0 | $\infty$ |
| Late | 0 | $\infty$ | 0 | $\infty$ |
| Night | 0 | 4 | 0 | 4 |
| Free | 0 | $\infty$ | 0 | 5 |

## F.1.2.2 Generation model

General parameters

| Property | Value |
| :--- | ---: |
| Max workdays after each other: | 4 |
| Allowed extra workdays per week: | 1 |
| Max workweekends after each other | 2 |
| Single workday allowed: | Yes |
| Min length free block: | 1 |

Shift block restrictions

| Shift | Block length |  |  | Days off |  |  |  |
| :--- | ---: | ---: | ---: | ---: | ---: | ---: | ---: |
|  |  | Bllowed block <br> Before |  | After |  | repetition |  |
|  | Min | Max | Min | Max | Min | Max |  |
| Day | 2 | 4 | 0 | $\infty$ | 0 | $\infty$ | 2 |
| Late | 1 | 3 | 0 | $\infty$ | 0 | $\infty$ | 0 |
| Night | 2 | 2 | 0 | $\infty$ | 2 | $\infty$ | 0 |

## Shift pattern allowance

| From |  | To | Always | Day off |
| :--- | :--- | :--- | :---: | :---: |
| Day | $\Rightarrow$ | Late | $\checkmark$ |  |
| Day | $\Rightarrow$ | Night | $\checkmark$ |  |
| Late | $\Rightarrow$ | Day |  | $\checkmark$ |
| Late | $\Rightarrow$ | Night | $\checkmark$ |  |
| Night | $\Rightarrow$ | Day |  | $\checkmark$ |
| Night | $\Rightarrow$ | Late |  | $\checkmark$ |

## F.1.2.3 Soft and hard constraints

## Weekly

| Day | Shift | Assign | Weeks | Counts demand | Type |
| :--- | :--- | :--- | :--- | :--- | :--- |
| Monday | Late | No | All | Yes | Generation |
| Friday | Day | Yes | All | No | Always |
| Saturday | Night | No | All | Yes | Generation |
| Sunday | Night | No | All | Yes | Generation |

## F.1.3 MC employee 1 (MC, Flexible, $20 \mathrm{hr} / \mathrm{wk}$ )

## F.1.3.1 Shift preferences

| Shift | Min count | Max count | Min length | Max length |
| :--- | ---: | ---: | ---: | ---: |
| Day | 0 | $\infty$ | 0 | $\infty$ |
| Late | 0 | $\infty$ | 0 | $\infty$ |
| Night | 0 | 4 | 0 | 4 |
| Free | 0 | $\infty$ | 0 | $\infty$ |

## F.1.3.2 Generation model

General parameters

| Property | Value |
| :--- | ---: |
| Max workdays after each other: | 5 |
| Allowed extra workdays per week: | 1 |
| Max workweekends after each other | 2 |
| Single workday allowed: | Yes |
| Min length free block: | 2 |

Shift block restrictions

| Shift | Block length |  | Days off |  |  |  | Allowed block repetition |
| :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: |
|  |  |  | Before |  | After |  |  |
|  | Min | Max | Min | Max | Min | Max |  |
| Day | 2 | 2 | 0 | $\infty$ | 0 | $\infty$ | $\infty$ |
| Late | 1 | 2 | 0 | $\infty$ | 0 | $\infty$ | 0 |
| Night | 2 | 4 | 0 | $\infty$ | 2 | $\infty$ | 0 |

## Shift pattern allowance

| From |  | To | Always | Day off |
| :--- | :--- | :--- | :---: | :---: |
| Day | $\Rightarrow$ | Late | $\checkmark$ |  |
| Late | $\Rightarrow$ | Day |  | $\checkmark$ |
| Late | $\Rightarrow$ | Night | $\checkmark$ |  |
| Night | $\Rightarrow$ | Day |  | $\checkmark$ |

## F.1.3.3 Soft and hard constraints

## Weekly

| Day | Shift | Assign | Weeks | Counts demand | Type |
| :--- | :--- | :--- | :--- | :--- | :--- |
| Monday | Late | No | All | Yes | Permanent |
| Tuesday | Late | No | All | Yes | Permanent |
| Wednesday | Late | No | All | Yes | Permanent |
| Thursday | Late | No | All | Yes | Permanent |
| Friday | Late | No | All | Yes | Permanent |

## F.1.4 MC employee 4 (MC, $36 \mathrm{hr} / \mathrm{wk}$ )

## F.1.4.1 Shift preferences

| Shift | Min count | Max count | Min length | Max length |
| :--- | ---: | ---: | ---: | ---: |
| Day | 0 | $\infty$ | 0 | $\infty$ |
| Late | 0 | 12 | 0 | $\infty$ |
| Night | 0 | 4 | 0 | 4 |
| Free | 0 | $\infty$ | 0 | $\infty$ |

## F.1.4.2 Generation model

General parameters

| Property | Value |
| :--- | ---: |
| Max workdays after each other: | 5 |
| Allowed extra workdays per week: | 1 |
| Max workweekends after each other | 2 |
| Single workday allowed: | No |
| Min length free block: | 2 |

Shift block restrictions

| Shift | Block length |  |  | Days off |  |  |  |
| :--- | ---: | ---: | ---: | ---: | ---: | ---: | ---: |
|  |  | Before |  | After <br> repet block |  |  |  |
|  | Min | Max | Min | Max | Min | Max |  |
| Day | 3 | 5 | 0 | $\infty$ | 0 | $\infty$ | 0 |
| Late | 1 | 2 | 0 | $\infty$ | 0 | $\infty$ | 0 |
| Night | 2 | 4 | 0 | $\infty$ | 2 | $\infty$ | 0 |

## Shift pattern allowance

| From |  | To | Always | Day off |
| :--- | :--- | :--- | :---: | :---: |
| Day | $\Rightarrow$ | Late | $\checkmark$ |  |
| Late | $\Rightarrow$ | Day |  | $\checkmark$ |
| Late | $\Rightarrow$ | Night | $\checkmark$ |  |
| Night | $\Rightarrow$ | Day |  | $\checkmark$ |

## F.1.4.3 Soft and hard constraints

There were no soft or hard constraints for this employee.

## F.1.5 MC employee 14 (MC, $28 \mathrm{hr} / \mathrm{wk}$ )

## F.1.5.1 Shift preferences

| Shift | Min count | Max count | Min length | Max length |
| :--- | ---: | ---: | ---: | ---: |
| Day | 0 | $\infty$ | 0 | $\infty$ |
| Late | 0 | $\infty$ | 0 | $\infty$ |
| Night | 0 | 4 | 0 | 4 |
| Free | 0 | $\infty$ | 0 | $\infty$ |

## F.1.5.2 Generation model

## General parameters

| Property | Value |
| :--- | ---: |
| Max workdays after each other: | 4 |
| Allowed extra workdays per week: | 1 |
| Max workweekends after each other | 2 |
| Single workday allowed: | No |
| Min length free block: | 2 |

Shift block restrictions

| Shift | Block length |  | Days off |  |  |  | Allowed block repetition |
| :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: |
|  |  |  | Before |  | After |  |  |
|  | Min | Max | Min | Max | Min | Max |  |
| Day | 2 | 5 | 1 | $\infty$ | 0 | $\infty$ | 0 |
| Late | 1 | 2 | 0 | $\infty$ | 0 | $\infty$ | 0 |
| Night | 2 | 4 | 0 | $\infty$ | 2 | $\infty$ | 0 |

## Shift pattern allowance

| From |  | To | Always | Day off |
| :--- | :--- | :--- | :---: | :---: |
| Day | $\Rightarrow$ | Late | $\checkmark$ |  |
| Late | $\Rightarrow$ | Day |  | $\checkmark$ |
| Late | $\Rightarrow$ | Night | $\checkmark$ |  |
| Night | $\Rightarrow$ | Day |  | $\checkmark$ |

## F.1.5.3 Soft and hard constraints

## Weekly

| Day | Shift | Assign | Weeks | Counts demand | Type |
| :--- | :--- | :--- | :--- | :--- | :--- |
| Tuesday | Late | No | All | Yes | Generation |
| Wednesday | Late | No | All | Yes | Generation |
| Friday | Free | No | All | Yes | Generation |

## F.1.6 STD employee 4 (STD, $36 \mathrm{hr} / \mathrm{wk}$ )

Remark: For this employee, we will only show part of the data. The employee has a study day on Tuesday and wants the Wednesday evening off.

## F.1.6.1 Generation model

## Shift block restrictions

| Shift | Block length |  | Days off |  |  |  | Allowed block repetition |
| :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: |
|  |  |  | Before |  | After |  |  |
|  | Min | Max | Min | Max | Min | Max |  |
| Day | 3 | 5 | 0 | $\infty$ | 0 | $\infty$ | 2 |
| Late | 1 | 2 | 0 | $\infty$ | 0 | $\infty$ | 0 |
| Night | 2 | 4 | 0 | $\infty$ | 2 | $\infty$ | 0 |

## F.1.6.2 Soft and hard constraints

Weekly

| Day | Shift | Assign | Weeks | Counts demand | Type |
| :--- | :--- | :--- | :--- | :--- | :--- |
| Tuesday | Day | Yes | All | No | Always |
| Wednesday | Late | No | All | Yes | Generation |

## F. 2 Parameter settings

This section will consider the general (cost) parameters used while generating rosters and solving the (I)LP.

## F.2.1 Cost functions for roster generation

These cost functions are used while generating rosters. At the end of the generation phase for a single employee, the roster with the lowest cost will retrieve the cost value 0 , the roster with the highest cost will retrieve the cost value 1 and the other rosters will retrieve a scaled inbetween value.

## F.2.1.1 Cost functions for deviating from the preferred shift block length

The concept is discussed in section 4.3.2.1.
The general idea is that a block which is longer than preferred may be more annoying than a block which is shorter than preferred. The costs of having a longer block of days off is high, since rosters may get very unbalanced when a few larger blocks of days off are allocated. The employee will then have more work shifts in another part of his roster.

The costs are defined as:

| Shift block | Shorter | Longer |
| :--- | ---: | ---: |
| Day | 2 | 4 |
| Late | 2 | 4 |
| Night | 3 | 6 |
| Free | 2 | 10 |

## F.2.1.2 Cost functions for deviating from the preferred shift count

The concept is discussed in section 4.3.2.2.
Again, the general idea is that having more unwanted shifts is worse than having less desired shifts. This holds especially for the sometimes very undesired night shift period. When such a preference is infringed, a small cost per day less/extra will be added for the employee. For a day off, this function does not make any sense, hence its value is 0 .

The costs are defined as:

| Shift type | Less | More |
| :--- | ---: | ---: |
| Day | 4 | 8 |
| Late | 4 | 8 |
| Night | 2 | 10 |
| Free | 0 | 0 |

## F.2.1.3 Other costs

The costs for having additional workdays during a week (a little unbalance) is defined as 10. Having an additional workday during a week is normal in this kind of work environments, but a nice spread of the workload is better.

However, the most important parts of soft constraints are the soft shift requirements (in case they are used). Allocating or infringing a soft shift constraint for a certain day will be considered as more important than the situations as described above. Therefore, these requirements will have a bigger influence on the total roster cost.

These costs are defined as:

- For each soft shift requirement which is satisfied, the cost is defined as -50 .
- For each soft shift requirement which is violated, the cost is defined as 100.

Allocation of a soft shift requirement is nice, so the total costs may get lower. However, violation of a soft shift requirement is annoying. Since there are 4 possible allocations for a day, it is likely another allocation is also possible. Hence the 'bonus' of satisfying a soft shift constraint will not replace the cost of infringing one.

## F.2.2 ILP cost functions

Although the costs of the selected employee rosters are taken into account in the linear program, the rosters to choose from are already quite acceptable for the employees, since the generated rosters with high costs will already be dropped before the ILP is solved. Therefore, the roster costs (scaled to values between 0 and 1) may remain very small, compared to the costs used to force a lower and upper bound on the number of employees in each shift.

The costs for satisfying the occupancy demands are designed in such a way that an overflow in a shift is always preferred over a shortness in another shift. Therefore, the costs for every employee short in a shift are very high and a solution containing it is very discouraged.
As seen in the ILP (chapter 3), three types of surplus will be distinguished: small surplus, big surplus and extreme surplus. The last one is needed for a better spread of employees during day shifts. For late and night shifts, the value does not make sense in realistic situations.
Furthermore, surplus is more preferred during weekdays than during weekends. Therefore, weekend costs have a higher value than the equal weekday costs.
Finally, the surplus during late and night shifts is considered as equal. Of course we could use higher values for night shifts (since a surplus in a night shift is extremely undersired), but since night shifts will always be allocated in blocks of at least 2 days, it is already more desired to occasionally allocate some employees to a single late shift instead of a night shift.

## F.2.2.1 Costs for shortness

For each employee or qualification shortage in a shift (parameter $z_{i}$ in the ILP), the cost is defined as 10000 .

## F.2.2.2 Costs for surplus

For the cost parameters $f_{i}, F_{i}$ and $F_{i}^{\prime}$ in the ILP, the value is defined as:

| Shift | Day | Small <br> $f_{i}$ | Big <br> $F_{i}$ | Extreme <br> $F_{i}^{\prime}$ |
| :--- | :--- | ---: | ---: | ---: |
| Day | Week | 0 | 10 | 410 |
|  | Weekend | 10 | 20 | 420 |
| Late | Week | 100 | 200 | 500 |
|  | Weekend | 200 | 400 | 600 |
| Night | Week | 100 | 200 | 500 |
|  | Weekend | 200 | 400 | 600 |

The values are a little bit arbitrarily chosen, but seemed to achieve the desired result: having as little shortness as possible and having all surplus spread over the day shifts.
For a day shift during weekdays, the minimum number of employees is 11 and the acceptable maximum number of employees is 14 . During weekends, this is 11 and 12 respectively. For a late and night shift, the acceptable maximum is equal to the minimum number of employees needed
in that shift. Initially, there also was a small 'acceptable' buffer for the late shifts, but this was changed since the UMC requested to have no surplus during late shifts also.

The small costs $\left(f_{i}\right)$ will be used for this 'acceptable' buffer. The big costs $\left(F_{i}\right)$ will be used for at most 2 employees above the acceptable buffer. The extreme costs ( $F_{i}^{\prime}$ ) will be used for every additional employee above this value.

## F. 3 The roster and demand satisfaction

The roster and the corresponding occupancy demand satisfaction are shown on the next pages. We will provide two rosters. The first roster shows the optimal solution for the period. The second roster also shows a nice solution, but this one was unfortunately calculated with a wrong mutation setting and we did not allow VK employee 2 to perform some night shifts. We still present this roster, because it can be compared to the repair results in appendix G.

For both rosters, the first page considers the individual roster for each employee. Each shift has its own color (day $=$ blue, late $=$ red, night $=$ orange, off $=$ green). Furthermore, a small red or green mark shows the hard demands during generation. A red mark in the corners shows the shift had to be allocated due to a hard generation or permanent constraint, a green mark shows that some shifts were forbidden for the day by a hard generation or permanent constraint.

The legends of the columns is the following:

- The column 'WD' shows the number of allocated work shifts during the six weeks period.
- The column 'Gen' shows whether an alternative generation scheme was used for generating the roster.

PR: This employee has a predefined roster (see the anonymized scheme on page 112).
PW: This employee has a roster based on predefined weeks (see the scheme on page 113).
AR: The employee may only have a schedule with 7 night shifts during in the first week and the next week off.

NU: This flexible ('nuluur') employee is currently not scheduled, but may obtain some shifts during the repair phase.
NG: The employee will not be scheduled during this period (e.g. due to a holiday, maternity leave, illness, etc...).

- The column ' 0 ?' shows the average number of hours per week a flexible employee is normally assigned. It should be noted that this number is something different than the numbers in the column WD.

The second page considers for each day, for each possible shift (day, late or night), for each possible qualification rule of the ILP the number of employees (which satisfy the qualification rule) which had to be assigned to the particular shift. Furthermore, it shows the number of employees that could not be assigned and thus are short in the shift. Also, the overflow of employees is shown.
The overflow could be used to see from which shifts some employees could be retracted in case a (manual) change of the roster is needed. It must however be noted that not every overflow of a certain qualification is really an overflow. In most cases, an employee with the particular qualification was assigned to satisfy the constraints for a lower qualification or the total employee demand. For instance, this frequently occurs during the late shifts.

## F.3.1 First solution (correct)

See page 131 for the explanation of the statement 'correct'.
The ILP was solved on a computer with an AMD Phenom X4 945 processor at 3.00 GHz . The size of the internal RAM memory was 12 GB .

For each employee, the calculation started with 100 rosters with lowest roster costs (whenever available). In the remaining part of this description, the 'best' roster for an employee will be defined as the roster with the lowest reduced cost. Of course an amount of rosters may only be added for an employee in case that amount is still available.

Each round, the 100 best rosters for each employee were added. At the end of the LP phase, the 2500 best rosters out of the set of rosters for all employees which were not added to the problem instance yet, were also added.

In the mutation phase all rosters that were selected in the solution of the LP were mutated and added to the problem instance.
Furthermore the 2500 best (out of the set of rosters for all employees) rosters which were added to the problem instance, but not used in the LP solution, were also mutated. The 2500 best mutations were also added to the problem instance. At most 1 mutation operations has been applied on each roster.
The LP phase was allowed to take 20 minutes and the ILP phase was allowed to take 15 minutes $^{2}$.

- The LP phase took 3 minutes and 3 seconds and was optimally solved.
- The ILP phase took 15 minutes and was cut off due to the time limit.
- The result of the LP phase before mutation is $72,996.31$
- The result of the LP phase after mutation is $69,230.07$
- The result of the ILP phase is $80,992.20$.
- The integrality gap is $4.48 \%$.

It should be noted that we also experienced longer times for the LP phase (approximately $10-15$ minutes) on some interim test inputs. Furthermore, the objective value may seem to be high, but this is logical since shortnesses already introduce a penalty of 10,000 .

[^26]






 い n $\boldsymbol{H} \mid \boldsymbol{N}$

乡 m｜N




 N
3 N










[^27]$\mid$ NIGHT mc needed
｜NIGHT mc short





｜NIGHT total short
｜NIGHT total overflow

## F.3.2 Second solution (some mistakes)

See page 131 for the explanation of the statement 'some mistakes'.
The ILP was solved on a computer with an AMD Phenom X4 945 processor at 3.00 GHz . The size of the internal RAM memory was 12 GB .
For each employee, the calculation started with 100 rosters with lowest roster costs (whenever available). In the remaining part of this description, the 'best' roster for an employee will be defined as the roster with the lowest reduced cost. Of course an amount of rosters may only be added for an employee in case that amount is still available.

Each round, the 100 best rosters for each employee were added. At the end of the LP phase, the 2500 best rosters out of the set of rosters for all employees which were not added to the problem instance yet, were also added.

In the mutation phase all rosters that were selected in the solution of the LP were mutated. The 5000 best unique mutations were added to the problem instance.

Furthermore the 2500 best (out of the set of rosters for all employees) rosters which were added to the problem instance, but not used in the LP solution, were also mutated. The 2500 best mutations of them were also added to the problem instance. At most 2 mutation operations have been applied on each roster.

The LP phase was allowed to take 20 minutes and the ILP phase was allowed to take 15 minutes $^{3}$.

- The LP phase took 2 minutes and 38 seconds and was optimally solved.
- The ILP phase took 15 minutes and was cut off due to the time limit.
- The result of the LP phase before mutation is $103,046.33$.
- The result of the LP phase after mutation is $97,020.54$.
- The result of the ILP phase is $121,037.64$.
- The integrality gap is $15.74 \%$.

It should be noted that we also experienced longer times for the LP phase (approximately $10-15$ minutes) on some interim test inputs. Furthermore, the objective value may seem to be high, but this is logical since shortnesses already introduce a penalty of 10,000 .

[^28]


 $\varkappa$ ~









 и о扣















## K． 己 0 0 0

い $\stackrel{\text { N }}{ }$






い！

M




u



з ！N ○ ○ $\operatorname{l}$








$\cdots \underset{\sim}{\omega}$


N










Month：

| Weekday： |
| :--- |
| Day： |
| DAY svk needed |
| DAY svk short |
| DAY svk overflow |
| DAY mc needed |
| DAY mc short |
| DAY mc overflow |
| DAY mc＿vk needed |
| DAY mc＿vk short |
| DAY mc＿vk overflow |
| DAY std needed short |
| DAY std overflow |
| DAY total needed |

AY total needed ATE svk needed

 ATE mc short LATE mc＿vk needed
 ATE std short ATE total needed ATE total short ATE total overflow NIGHT svk short NIGHT mc needed NIGHT mc short
 NIGHT mc＿vk needed
NIGHT mc＿vk short IGHT std needed NIGHT std short NIGHT total needed
 NIGHT total overflow

## Appendix G

## Results of the repair phase

This appendix contains the results of the experiments with the repair heuristics.

## G. 1 Basic repairing

The next page will show the results of automatically repairing the rosters defined in appendix F. The first presented roster (page 139) is the repaired version of the roster on page 133. The second presented roster (page 141) is the repaired version of the roster on page 136.


 ц N







 з 0







玉 扣










 N

з i人」


 に


 $\vdash$ ォ


Month：









ल



$\Sigma$ に




$\vdash$ に

n N


క N|N





ธ










Month:










 に

 з

















 N







 $\vdash$ ~
 Month:
ㅇ
岂
0
0
3
u ท
น N
$\vdash \vec{N}$
U N

$$
\Sigma N
$$

$$
\text { u } \stackrel{m}{N}
$$

$$
N 3
$$

$$
\vdash 9
$$

$$
\Sigma \stackrel{\infty}{1}
$$

u


## G. 2 Advanced repairing

The next pages will show the results of the additional repair experiments with the roster defined on page 133 in appendix F. The experiment setup could be found in section 6.8.2.

The schedule on page 144 shows the result when it is allowed to assign shifts to flexible employees. The schedule on page 146 shows the result when this is not allowed.


 m


























 N










$$
\underset{N}{N}
$$









WD GEN 0?





































 Dav: DAY svk needed DAY svk nek short DAY svk overlow DAY mc needed DAY mc short DAY mc overflow DAY mc_vk needed DAY mc_vk overf DAY std needed DAY std short DAY total needed DAY total short DAY total overflow LATE svk needed LATE svk short LATE svk overflow | $t$ |
| :--- |
| 4 |
| 4 | LATE mc overflow LATE mc_vk needed LATE mc_vk ne_vk short LATE std needed LATE std short LATE std overflow |LATE total needed LATE total short

LATE total overflow NIGHT svk needed



NIGHT mc overflow NIGHT mc_vk needed
NIGHT mc_vk short


NIGHT std short

 |NIGHT total short
|NIGHT total overflo


[^0]:    ${ }^{1}$ The unit direction calls the appointment form 'zero-hour based', but the fashion in which it is used appears more like a 'call contract with a prior made agreement'[26]. However, the exact contract form is not relevant here.

[^1]:    ${ }^{2}$ A fast cycle, hence all shift blocks are short. A particular shift is allocated on a few consecutive days only instead of during a full week.

[^2]:    ${ }^{3}$ Unfortunately, there will be a deviation of these rules in rare cases.
    ${ }^{4}$ The night between Monday and Tuesday and the night between Tuesday and Wednesday.

[^3]:    ${ }^{5}$ According to the collective bargaining agreement ( $C B A$ ) (see appendix A), every employee must have a period of 36 hours per week or 72 hours per two weeks (possibly to be split up in parts of at least 32 hours) for which he has no duties. This is already satisfied when a shift pattern like Day $\rightarrow$ Free $\rightarrow$ Day or Day $\rightarrow$ Free $\rightarrow$ Late occurs in a certain week. Since forwards rotating rosters are preferred and the biggest appointments are 4.5 days per week, these rules are automatically met. Hence the CBA rule can be simplified.

[^4]:    ${ }^{6}$ There is no clear definition for a week. Judging from example schedules, a week can be defined as Monday Sunday. An interesting alternative is to use Sunday - Saturday, since it makes it easier to allow days off during the (start of the) week before a weekend off.

[^5]:    ${ }^{7}$ Note that an SVK Nurse with a 'quality day' is not allowed to count for these demands.

[^6]:    ${ }^{1}$ Unfortunately, the paper is not clear whether this solution is completely based on the shift patterns, although it is likely the case.

[^7]:    ${ }^{2}$ http://www-01.ibm.com/software/integration/optimization/cplex-optimizer/
    ${ }^{3}$ An LP solution approach, without using techniques such as revised simplex and column generation.
    ${ }^{4} \mathrm{~A}$ pattern is for instance a roster of an employee.

[^8]:    ${ }^{5}$ Since the fractional knapsack problem is a quite 'simple' problem, there exists a greedy heuristic which produces the optimal total value. This can be achieved by sorting all items on decreasing $\frac{c_{j}}{w_{j}}$ value and then taking all items with the highest proportion until the bag is full. For the examples, we suppose no such heuristic exists.

[^9]:    ${ }^{6}$ Examples: An early shift every Wednesday, never a late shift on Tuesday or preferences conflicting with the rotation parameters, such as a day off on every Wednesday and Friday, but working on every Thursday.
    ${ }^{7}$ Rosters which allow for example a late shift every Monday and a day shift every Wednesday.

[^10]:    ${ }^{1}$ See section 3.1.6 for an explanation of this construction.

[^11]:    ${ }^{2}$ Although we can use it to represent unqualified staff, we do not need it in our solution.

[^12]:    ${ }^{3}$ Since it is allowed to add 2 students to obtain an occupancy of 11 employees, we need only $11-2=9$ people with an MC- and/or VK-qualification in total, as long as the MC-qualification demand of 5 is met.

[^13]:    ${ }^{4}$ Naturally, it is possible to have a different cost parameter $g_{i}^{q}$ for every qualification shortness parameter, but for simplicity we decided to use a single cost parameter. To stimulate a better dispersion of employees over the shifts when the amount of available personnel in a roster period is substantial larger than the demands, it is also possible to add a third surplus parameter $Y_{i}^{\prime}$ with a cost value $F_{i}^{\prime}\left(>F_{i}\right)$. These parameters will then prevent having some shifts with extreme deviations of the average number of employees in similar shifts.

[^14]:    ${ }^{1} k=6$ in practice, but the method works for any number of full weeks.

[^15]:    ${ }^{2}$ Later, the UMC stated that the 'preferences' for most employees were internally also treated as rather hard constraints and should preferably be treated this way by the algorithm too.

[^16]:    ${ }^{3}$ The night between Monday and Tuesday and the night between Tuesday and Wednesday.

[^17]:    ${ }^{4}$ Currently, the mutation function for predefined rosters is not supported.

[^18]:    ${ }^{1}$ The cost settings in the ILP were comparable to the ones defined in appendix F .

[^19]:    ${ }^{2} a_{k}$ denotes the number of workdays during $k$ weeks, see subsection 4.1.2.
    ${ }^{3}$ We could not guarantee this for small holidays, since allocation of less workshifts may trigger some bounding rules less rapidly.

[^20]:    ${ }^{1}$ By default, this is $k$ times the appointment size of the employee for a period of $k$ weeks. There might be a deviation of one workday. When the employee has a predefined schedule, then the number of workdays from this schedule is taken.

[^21]:    ${ }^{2}$ See the exceptions below.

[^22]:    ${ }^{3}$ The number 2 is a little bit arbitrarily chosen. Flexible employees may work during the weekends, but they also should not obtain too many weekend shifts.

[^23]:    ${ }^{4}$ In case a late shift has to be assigned, an employee with less late shifts will be preferred over an employee with more late shifts. For a day shift, it is the other way around. However, we could also take the individual preferences for day and late shifts into account in order to provide more late shifts to employees who like them.

[^24]:    Legend

    * $=$ Only the best 2,500 have been added as a column in the ILP.
    ** $=$ Maximization: the 5,000 best schedules (out of 45,852 ).

[^25]:    ${ }^{1}$ When the value is 1 , both $x \rightarrow$ Free $\rightarrow x$ and $x \rightarrow$ Free $\rightarrow y(y \neq x)$ are possible, but the situation $x \rightarrow$ Free $\rightarrow x \rightarrow$ Free $\rightarrow x$ is not possible. When the value is 2 , the latter case is also possible.

[^26]:    ${ }^{2}$ Unfortunately, the computer experienced some heat problems when it was allowed to solve the ILP for 60 minutes.

[^27]:    
    LATE total needed
    ｜LATE total short
    
    
    

[^28]:    ${ }^{3}$ Unfortunately, the computer experienced some heat problems when it was allowed to solve the ILP for 60 minutes.

