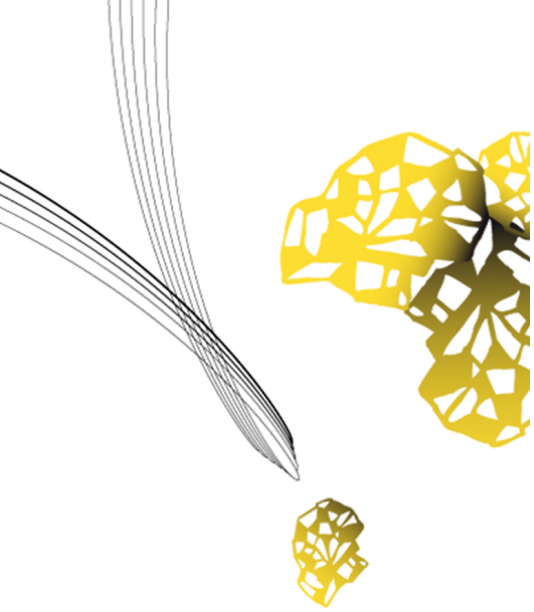


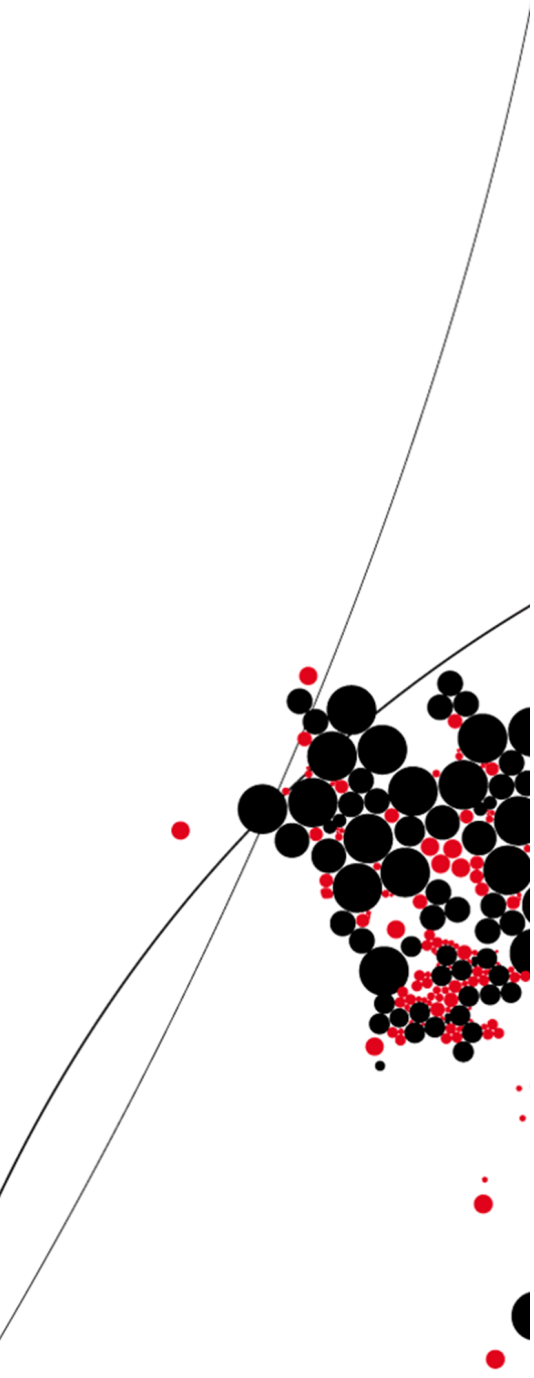
UNIVERSITY OF TWENTE.

Discrete Mathematics and Mathematical Programming

Master's Thesis

A Two-Phase Approach to the Shifts and Breaks Design Problem Using Integer Linear Programming

A.B.P. Akkermans



Supervisors
Dr.ir. G.F. Post (UT)
Prof.dr. M.J. Uetz (UT)

Graduation Committee
Dr.ir. G.F. Post (UT)
Prof.dr. M.J Uetz (UT)
Dr. J.C.W. van Ommeren (UT)

Enschede,
November 17, 2017

Abstract

In this thesis we make use of integer linear programming methods to obtain solutions to the shift design problem, the break scheduling problem, and the shifts and breaks design problem. Results are obtained by using the commercially available optimisation package Cplex. By using integer linear programming we are able to proof optimal solutions for many instances of shift design which were not known before. Furthermore this approach shows better results than existing methods when allowing only a short running time. Our approach to break scheduling shows not to be competitive with results in the literature, however it shows to be effective in our two-phased approach to the shifts and break design problem.

We combine our approaches used for shift design and break scheduling to form a two-phased approach to the shifts and breaks design problem. This two-phased approach out performs the current best method for shifts and breaks design on a set of randomly generated instances as well as on a set of real life instances.

Contents

1 Introduction

Personnel scheduling was first introduced by Edie [17] and formulated as a set covering problem by Dantzig [11] in the 1950's. After its introduction it has received a great deal of attention in the literature and has been applied to numerous different areas such as airlines, health care systems, police, call centres and retail stores [18]. The interest can be explained by labour cost being a major direct cost component for companies.

In this thesis we will consider the shifts and breaks design problem [15] which combines the shift design problem [28] and the break scheduling problem [6]. The shift design problem is a variation on the personnel scheduling problem and was first introduced by Musliu et al. [28]. The problem arose in a project involving members of Vienna University of Technology and Ximes Corp., a company offering software and consulting services regarding working hours. Based on the forecasted number of employees needed during each time period of a planning horizon, a set of shifts are designed and a number of employees (duties) are assigned to the shifts for each day. A goal in the shift design problem is to have a low number of shifts. These shifts can be re-used over multiple days and this makes the problem different from other personnel scheduling problems discussed in the literature. Schedules with a low number of shifts are easier to read and manage, and make employee scheduling easier such that groups of people stay together.

The break scheduling problem, as we will consider here, was first considered by Beer et al. [6]. Their solution methods were applied to real life instances of supervisory personnel. For this problem a set of duties are given where each duty represents a set of consecutive time slots in which an employee will be present. Breaks need to be scheduled for each duty such that many conditions are satisfied and the forecasted staffing requirements are closely met. Supervisory personnel spends most of their time in front of computer monitors and it is required to keep a high level of concentration throughout the day. Therefore the problem formulation requires many small breaks as well as a lunch break somewhere in the middle of the shift. The large number of break possibilities produce a complex problem and separates the break scheduling problem from other problems discussed in the literature.

The shifts and breaks design problem combines the two problems mentioned above. The goal is to design shifts and breaks for each duty such that the staffing requirements are closely followed and the number of shifts used is small. Tackling this combined problem was first done by Di Gaspero et al. [15]. In their formulation a large number of breaks as well as a large number of possible shifts are allowed and as a consequence the search space is enormous.

Solution methods to the shift design and the break scheduling problems are part of a commercial product called Operating Hours Assistant [35].

1.1 Aim of the Master's Thesis

In this thesis we will investigate the shifts and breaks design problem and propose a solution approach to improve current best found solutions. In our solution approach we split the shifts and breaks design problem into two different phases and use integer linear programming to find solutions for each phase. The first phase is similar to the shift design problem and the second phase in an instance of break scheduling. We use parts of our solution approach to compare the effectiveness of integer linear programming for the shift design problem and the break scheduling problem, as compared to local search methods proposed in the literature. In short the three goals of this thesis are as follows.

1. Propose a solution approach to the shifts and breaks design problem to improve solutions found by Di Gaspero et al. [15]

2. Compare the effectiveness of integer linear programming methods on instances of shift design as compared to local search methods used in the literature.

3. Compare the effectiveness of integer linear programming methods on instances of break scheduling as compared to local search methods used in the literature.

1.2 Results of the Master's Thesis

Integer linear programming shows to be effective on instances of shift design, both in proving optimal solutions and in finding good quality solutions quickly. For break scheduling our approach using integer linear programming shows not to be competitive with other approaches available in the literature. Our two-phased approach to the shifts and breaks design problem shows to outperform the current best available method.

1.3 Structure of the Master's Thesis

The remaining chapters of this thesis are organized as follows.

Chapter 2 describes the shift design problem, the break scheduling problem, and the shifts and breaks design problem.

Chapter 3 gives an overview of literature regarding personnel scheduling with a special focus towards shift design, break scheduling, and shifts and breaks design.

Chapter 4 shows our solution approach and results for the shift design problem.

Chapter 5 shows our solution approach and results for the break scheduling problem.

Chapter 6 shows our solution approach and results for the shifts and breaks design problem.

Chapter 7 summarises our results and gives suggestions for future research.

2 Problem Descriptions

In this section we will describe the three problems discussed in this thesis, the shift design problem, the break scheduling problem, and the shifts and breaks design problem. For each problem we will start with an introduction and give a formal description afterwards.

2.1 Shift Design

In this section we will first introduce the shift design problem and give a give a formal description afterwards. The problem is formulated as described by Di Gaspero et al. [14].

2.1.1 Introduction

The shift design problem considers the design of shifts and assigning a number of workers to each shift such that the number of workers present over the planning horizon closely aligns with the workforce requirements. The planning horizon consists of multiple days which are split up into n equally long consecutive time periods. The desired staffing level, *requirements*, for each time period are given. Furthermore the problem is considered to be cyclic, which means that employees starting on the last day and working overnight will be considered to be active from the first time period onwards. Figure 1 shows an example of staffing requirements spread out over a two day cycle.

The requirements state how many duties are necessary in order to cope with the demand at any time period and therefore it is desired to have at least as many employees working as the requirements. On the other hand having too many employees working is a waste of resources. Therefore it is desired to follow the requirements closely. In order to measure deviations from the requirements we will use the terms overstaffing and understaffing. Overstaffing counts the surplus in number of employees over the staffing requirements and understaffing counts the deficit. Both overstaffing and understaffing are allowed in the shift design problem and different importance can be put into these two factors.

In shift design it is also desirable to have a small number of shifts. In the context of shift design (and this thesis!) a *shift* is determined by its starting time, ranging from 00:00 to 23:59, and its length. A shift is considered to be active on all days of the planning horizon and can be assigned a different number of workers on each day. An example of a shift would be a morning shift starting at 06:00 and lasting 8 hours. Employees can be assigned to this shift on each day, thus it is possible to have a different number of employees on this shift on each day. More specifically, for each shift in the shift design problem we assign a number of *duties* for each day in the planning horizon. The term duty is chosen because it represents a workload which has to be fulfilled by a single employee. Hence the decisions made in shift design are designing the shifts and assigning a number of duties to each shift for each day. There are various reasons for wanting to have a small number of distinct

Figure 1: Example Staffing Requirements

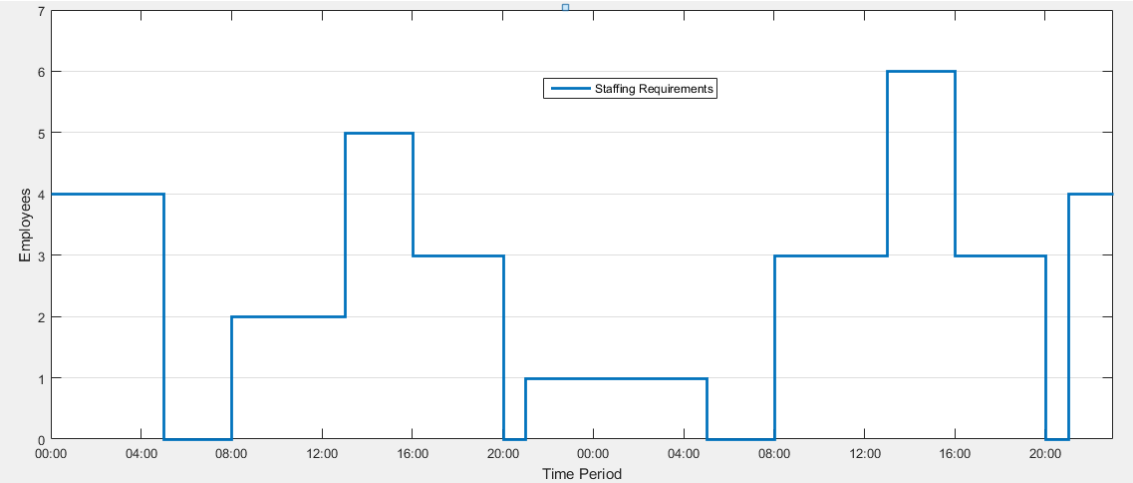


Table 1: Example Shift Types

Abbr.	Name	Earliest start	Latest start	Minimum length	Maximum length
M	Morning Shift	05:00	08:00	7:00	9:00
D	Day Shift	09:00	11:00	7:00	9:00
E	Evening Shift	13:00	15:00	7:00	9:00
N	Night Shift	21:00	23:00	7:00	9:00

shifts. As stated by Di Gaspero et al. [14] : "once a shift is selected (at least one person works in this shift during any day) it is neither really important how many persons work at this shift, nor on how many days the shift is reused. Nevertheless, it is important to have only few shifts as they lead to schedules that have a number of advantages. For example, they are more adequate if one wants to keep teams of persons together, whenever this is necessary because of managerial or qualification reasons. Besides this, there are further advantages of obtaining schedules with fewer shifts. For example, they lead to schedules that are easier to design with or without software support [27]. Fewer shifts also make schedules easier to read, check, manage and administer, being each of these activities a burden in itself."

The shifts can only be chosen based on a set of *shift types*. A shift type specifies the earliest and latest starting time of a shift, as well as the shortest and longest duration of a shift. In Table 1 an example of possible shift types are shown. Using these shift types we can design three different shifts and a number of duties for each shift on each of the two days to exactly meet the requirements from Figure 1. This solution is shown in Table 2 and Figure 2.

Figure 2: Example Solution Graph

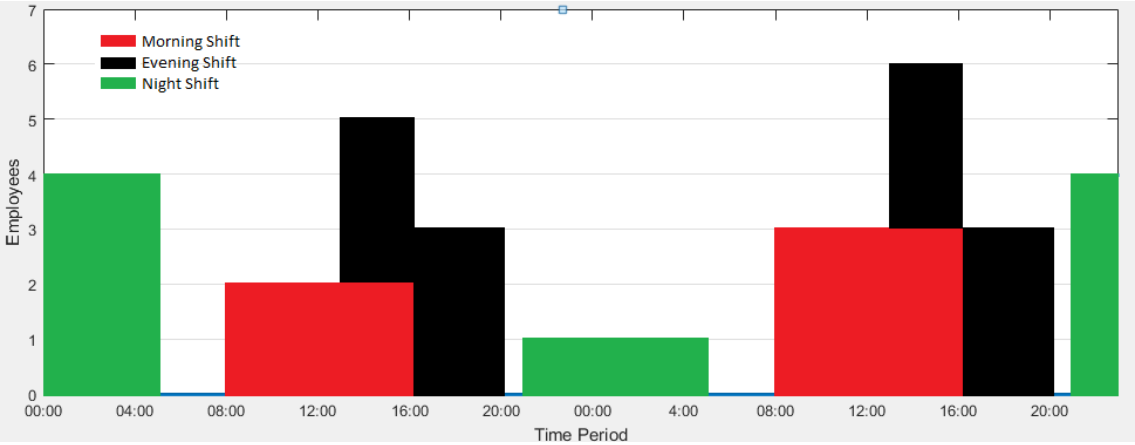


Table 2: Example Solution Table

Shift Type	Start	End	Duties Day 1	Duties Day 2
Morning	08:00	16:00	2	3
Evening	13:00	20:00	3	3
Night	21:00	05:00	1	4

2.1.2 Formal Description

In this section we give a formal description of the shift design problem. First the factors describing an instance of shift design are given. Afterwards the decision variables are described. Finally we show the objective of the problem.

Instance Description
An instance of shift design can be described by the following.

- D consecutive days.
- These days are spanned by n equally long consecutive time slots $T = \{t_1, t_2, ..., t_n\}$, The problem is cyclic hence t_1 follows t_n.
- Staffing requirements $R_t \quad \forall t \in T$.
- Set of shift types Y, each shift type, $y \in Y$contains the minimum and maximum start and duration of a shift of that type:
  $y.\text{min\_start}, y.\text{max\_start}, y.\text{min\_length}, y.\text{max\_length}$.
- W_1, W_2 and W_3 representing respectively the penalty for overstaffing, understaffing, and the number of shifts.

Decision Variables
The decision variables in the shift design problem are the following.

- A set of shifts S, where for each $s \in S$ we choose the type, start and length, $s.type$, $s.start$ and $s.length$.
  The start and length of each shift need to comply with the shift type hence we require

$$\begin{aligned} &s.type.\text{min\_start} && \leq s.start && \leq s.type.\text{max\_start} && \forall s \in S \\ &s.type.\text{min\_length} && \leq s.length && \leq s.type.\text{max\_length} && \forall s \in S \end{aligned}$$

- For each shift the number of duties at each day
  $w_{s,d} \quad \forall s \in S, d \in \{1, ..., D\}$

Objective
The objective is made up of three parts: the overstaffing, the understaffing and the number of shifts. In order to calculate the overstaffing and understaffing we define the *active workers* for each time period represented by $a_t \quad \forall t \in T$. a_t will be equal to the number of employees working at time period t.

$$a_t = \sum_{s \in S} x_{s,d}, \text{ where } x_{s,d} = \begin{cases} w_{s,d} & \text{if time slot } t \text{ belongs to the interval of shift } s \text{ on day } d \\ 0 & \text{otherwise} \end{cases}$$

We will use O to denote the total overstaffing and U to denote the total understaffing. We define them as follows.

$$O = \sum_{t \in T} max\{a_t - R_t, 0\}$$
$$U = \sum_{t \in T} max\{R_t - a_t, 0\}$$

The objective is to minimise the weighted sum of overstaffing, understaffing and the number of shifts.

$$\text{minimize } W_1 \cdot O + W_2 \cdot U + W_3 \cdot |S| \tag{1}$$

2.2 Break Scheduling

First we will give an introduction to the break scheduling problem. The formulation that we use is taken from [33]. The break scheduling problem uses characteristics of the shift design problem and we use similar notation as in the shift design problem. Therefore we will describe break scheduling as the problem of scheduling breaks for each duty as opposed to breaks for a shift. This will allow us to smoothly combine the two separate problems to formulate the shifts and breaks design problem afterwards.

2.2.1 Introduction

The break scheduling problem considers the allocation of breaks to a workforce. In this problem the planned shifts and number of employees on each day are considered to be fixed. In the break scheduling problem we are deciding on which breaks to allocate for each duty. A break can be characterised by its starting time and the length. Various restrictions are set on the breaks. These restrictions are usually set to ensure that workers can function optimally and are based on labour rules and therefore must be satisfied.

A duty is made up of two complementary time periods namely the breaks and the *working periods*. Therefore the working periods represent the time periods of a duty in which the employee is continuously working. We do however not consider a duty to be *active* on the first time period following a break. Instead we consider the duty to be getting accustomed with a new working situation after their break. Hence the first time period following a break does not count towards the break time of a duty and neither to the staffing requirements.

In Figure 3 an example of the staffing requirements and the *present workers* of an instance of break scheduling are shown. The present workers are the number duties who are working without accounting for breaks. Table 3 shows the duties used in this example, as well as a break allocation for each duty which will result in a perfect coverage i.e. no overstaffing nor understaffing. It should be noted that this example was constructed such that exact coverage was possible, it is not guaranteed that such a solution exists.

Figure 3: Example Staffing Requirements

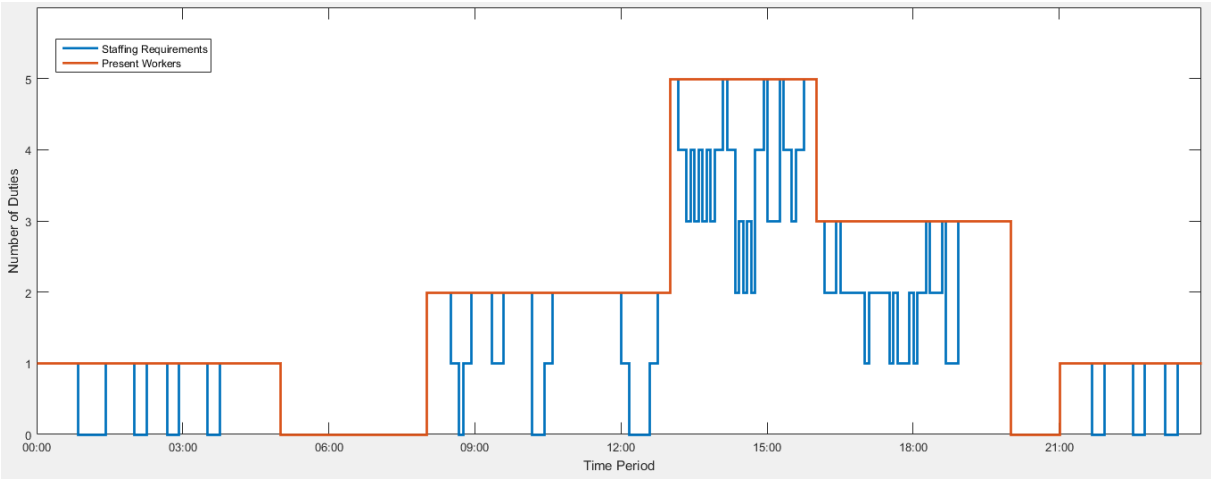


Table 3: Example Duties and Breaks

Duty	Start	End	Breaks (Start, Length)						
1	08:00	16:00	08:40, 10	10:10, 20	12:10, 30	13:20, 10	14:10, 10	15:00, 10	
2	08:00	16:00	08:30, 10	09:20, 10	10:10, 10	12:00, 30	13:10, 10	13:50, 10	14:40, 10
3	13:00	20:00	13:30, 10	14:20, 10	15:00, 10	16:30, 30	17:40, 10	18:20, 10	
4	13:00	20:00	13:40, 10	14:30, 10	15:20, 10	16:10, 10	17:30, 30	18:40, 10	
5	13:00	20:00	14:40, 20	15:30, 10	17:00, 30	18:00, 10	18:40, 10		
6	21:00	05:00	21:40, 10	22:30, 10	23:10, 10	00:50, 30	02:00, 10	02:40, 10	03:30, 10

The restriction on breaks and working periods are as follows.

- Total break length is fixed (depends on the length of the duty).
- Breaks have a minimum and maximum length.
- Working periods have a minimum and maximum length.
- If the working periods is longer than *long working period* the next break has a higher minimum length.
- Break cannot start too early or too late for a duty so that employees is always working in the first few and last few time periods of their duty.
- Duties which last longer than *minimum length for lunch* need to be allocated a lunch break. This lunch break has a fixed length and needs to positioned be around the middle of the duty.

2.2.2 Formal Description

In this section we give a formal description of the break scheduling problem. First we give the factors used to describe an instance of break scheduling. Afterwards the decision variables are described. Next the restrictions on feasible break patterns are specified. Finally we show the objective of the problem.

Instance Description
An instance of break scheduling can be described by the following.

- A set of n equally long consecutive time slots $T = \{t_1, t_2, ..., t_n\}$,
  The problem is cyclic hence t_1 follows t_n.
- Staffing requirements $R_t \quad \forall t$.
- Set of duties E, Each duty e has a starting time slot and a length,
  $e.start$, and $e.length$.
- W_1 and W_2 representing respectively the penalty for overstaffing and understaffing.
- Parameters for the constraints:

  total_break_time (for each duty)
  break_minimum_length and **break_maximum_length**
  working_period_minimum_length and **working_period_maximum_length**
  long_working_period and **long_break_minimum_length**
  earliest_break_start and **latest_break_start**
  min_length_for_lunch, **lunch_break_length**,
  earliest_lunch_break_start and **latest_lunch_break_start**.

Decision Variables
For each duty we are deciding on the number of breaks, and for each break on the start and length of the break. We let $e.breaks$ denote the number of breaks of duty e. Furthermore we let $b_{e,i}$ refer to the ith break of duty e.
Then the decision variables are

- $e.breaks \quad \forall e \in E$
- $b_{e,i}.start \quad \forall i \in \{1, e.start\}, e \in E$
- $b_{e,i}.length \quad \forall i \in \{1, e.start\}, e \in E$

$b_{e,i}.start$ denotes the start of the break relative to the start of duty e, i.e. $b_{e,i} = 1$ implies that the ith break of duty e starts on the 1st time period of duty e.
These variables also determine the working periods. we will use $wp_{e,i}$ to denote the length of the ith working period of duty e. If duty e contains $e.breaks$ breaks it contains $e.breaks+1$ working periods.

Restrictions
The total break time of each duty is given, we will use $e.$**total_break_time** to denote this for duty e. In order to ensure exactly this amount of break time is allocated we require the following constraints.

$$\sum_{i=1}^{e.breaks} b_{e,i}.length = e.\textbf{total\_break\_time} \quad \forall e \in E \tag{2}$$

A break has a minimum and a maximum length. The following constraints ensure this.

$$\textbf{break\_minimum\_length} \leq b_{i,e}.length \quad \forall i = 1, ..., e.breaks, e \in E \tag{3}$$
$$\textbf{break\_maximum\_length} \geq b_{i,e}.length \quad \forall i = 1, ..., e.breaks, e \in E \tag{4}$$

Similarly, the working periods have a minimum and maximum length.

$$\textbf{working\_period\_minimum\_length} \leq wp_{e,i} \quad \forall i = 1, ..., e.breaks + 1, e \in E \tag{5}$$
$$\textbf{working\_period\_maximum\_length} \geq wp_{e,i} \quad \forall i = 1, ..., e.breaks + 1, e \in E \tag{6}$$

Breaks following a long working period have a different minimum length. We require this by the following implication.

$$\begin{aligned} wp_{e,i} \geq \textbf{long\_working\_period} \implies & \\ b_{e,i} \geq \textbf{long\_break\_minimum\_length} \quad & \forall i = 1, ..., e.breaks, e \in E \end{aligned} \tag{7}$$

Breaks have an earliest and latest possible starting time.

$$b_{e,i}.start \geq \textbf{earliest\_break\_start} \quad \forall i = 1, ..., e.breaks, e \in E \tag{8}$$
$$b_{e,i}.start \leq \textbf{latest\_break\_start} \quad \forall i = 1, ..., e.breaks, e \in E \tag{9}$$

In case that the length of a duty is longer than the minimum length required for a lunch break, one of the breaks needs to be assigned the status of lunch break. This break has a fixed length and bounds on the earliest and latest start.

$$\begin{aligned} e.length \geq \textbf{min\_length\_for\_lunch} \quad & \implies \exists i \quad s.t. \\ b_{e,i}.start \geq \textbf{earliest\_lunch\_break\_start} \quad & \wedge \\ b_{e,i}.start \leq \textbf{latest\_lunch\_break\_start} \quad & \wedge \\ b_{e,i}.length = \textbf{lunch\_break\_length} \quad & \forall e \in E \end{aligned} \tag{10}$$

Objective

In the break scheduling problem the objective is made up of two parts: the overstaffing and the understaffing. In this section we define the *active workers* again which has a different meaning as compared to the shift design problem. However the implication of the active workers remains: it is the number of employee who are present and actively working at each time period.

$$a_t = \sum_{e \in E} x_{t,e}, \text{ where } x_{t,e} = \begin{cases} 1 & \text{if time slot } t \text{ belongs to the interval of duty } e \text{ and} \\ & \text{the duty is not on break on period t or period t-1} \\ 0 & \text{otherwise} \end{cases}$$

A duty is not active on the time period after a break since we assumed that this time period is used to acclimatize to new working conditions after returning from a break.

As in the shift design problem we will use O to denote the total overstaffing and U to denote the total understaffing. We define them as follows.

$$O = \sum_{t \in T} max\{a_t - R_t, 0\}$$
$$U = \sum_{t \in T} max\{R_t - a_t, 0\}$$

The objective is to minimise the weighted sum of overstaffing and understaffing.

$$\text{minimize } W_1 \cdot O + W_2 \cdot U \tag{11}$$

2.3 Shifts and Breaks Design

In this section we start with an introduction to the shifts and breaks design problem. Afterwards we will give a formal description of the problem. We follow the formulation used by [15].

2.3.1 Introduction

The shifts and breaks design problem combines the shift design and the break scheduling problem. The staffing requirements over a planning horizon are given. The goal of the problem is to find a small set of shifts with a number of duties allocated to each shift on each day, and a break allocation to each duty, such that the staffing requirements are closely followed.

The shifts can be chosen from a set of shift types as outlined in the shift design problem. There are many restrictions set on a feasible break allocation as is described in the break scheduling problem.

2.3.2 Formal Description

First we will list the factors which describe an instance of shifts and breaks design. Afterwards we define the decision variables of the problem. In the next section we will state the restrictions on the decision variables. Finally we give the objective function.

Instance Description

- D Consecutive days.
- These days are spanned by n equally long consecutive time slots $T = \{t_1, t_2, ..., t_n\}$, The problem is cyclic hence t_1 follows t_n.
- Staffing requirements $R_t \quad \forall t$.
- Shift types $y \in Y$, each shift type contains the minimum and maximum length and duration of a shift of that type:
  $y.\text{min\_start}, y.\text{max\_start}, y.\text{min\_length}, y.\text{max\_length}$.
- W_1, W_2 and W_3 representing respectively the penalty for
  overstaffing, understaffing, and the number of shifts.
- a function $f(s)$ which gives the **total_break_time** for shift s.

- parameters for the constraints on breaks:
  break_minimum_length and **break_maximum_length**
  working_period_minimum_length and **working_period_maximum_length**
  long_working_period and **long_break_minimum_length**
  earliest_break_start and **latest_break_start**
  min_length_for_lunch, **lunch_break_length**,
  earliest_lunch_break_start and **latest_lunch_break_start**.

Decision Variables
In this problem we are designing the shifts. For each shift we decide on the number of duties it will have on each day. Furthermore for each of these duties we decide on their break allocation. We use the following decision variables regarding the design of the shifts.

- A set of shifts S, where for each shift $s \in S$ we decide on the type, start and length, $s.type, s.start$ and $s.length$.
- For each shift the number of duties at each day,
  $w_{s,d} \quad \forall s \in S, d \in \{1, ..., D\}$.

We will use $s_{d,e}.breaks$ to denote the number of breaks for duty e starting on day d of shift s. Furthermore we let $b_{s,d,e,i}$ refer to the ith break of duty e starting on day d of shift s. Then the decision variables regarding the breaks are the following.

- $s_{d,e}.breaks \quad \forall s \in S, d \in \{1, ..., D\}, e \in \{1, ..., w_{s,d}\}$
- $b_{s,d,e,i}.start \quad \forall s \in S, d \in \{1, ..., D\}, e \in \{1, ..., w_{s,d}\}, i \in \{1, ..., s_{d,e}.breaks\}$
- $b_{s,d,e,i}.length \quad \forall s \in S, d \in \{1, ..., D\}, e \in \{1, ..., w_{s,d}\}, i \in \{1, ..., s_{d,e}.breaks\}$

Restrictions
The shifts need to be of a type as defined in the input and the start and length of each shift need to comply with its type. Hence we require the following constraints.

$$s.type.\text{min\_start} \leq s.start \leq s.type.\text{max\_start} \qquad \forall s \in S$$
$$s.type.\text{min\_length} \leq s.length \leq s.type.\text{max\_length} \qquad \forall s \in S$$

Furthermore a set of restrictions is placed on the break allocation of each duty. The total break time of each shift is given by $f(s)$ and the break time allocated to each duty of this shift needs to be equal to this.

$$\sum_{i=1}^{e_{s,d}.breaks} b_{s,d,e,i}.length = f(s) \quad \forall s \in S, d \in \{1, ..., D\}, e \in \{1, ..., w_{s,d}\} \tag{12}$$

The following constraints need to hold for all breaks, which means the constraints should hold $\forall s \in S, d \in \{1, ..., D\}, e \in \{1, ..., w_{s,d}\}, i \in \{1, ..., e_{s,d}.breaks\}$

$$b_{s,d,e,i}.length \geq \textbf{break\_minimum\_length} \tag{13}$$
$$b_{s,d,e,i}.length \leq \textbf{break\_maximum\_length} \tag{14}$$
$$wp_{s,d,e,i} \geq \textbf{working\_period\_minimum\_length} \tag{15}$$
$$wp_{s,d,e,i} \leq \textbf{working\_period\_maximum\_length} \tag{16}$$
$$wp_{s,d,e,i} \geq \textbf{long\_working\_period} \implies$$
$$b_{s,d,e,i} \geq \textbf{long\_break\_minimum\_length} \tag{17}$$
$$b_{s,de,i}.start \geq \textbf{earliest\_break\_start} \tag{18}$$
$$b_{s,d,e,i}.start \leq \textbf{latest\_break\_start} \tag{19}$$
$$\tag{20}$$

Lastly, for each shift of sufficient length, all duties need to be allocated a lunch break.

$$\begin{aligned} &e.length \geq \textbf{min\_length\_for\_lunch} && \implies \exists i s.t. \\ &b_{s,d,e,i}.start \geq \textbf{earliest\_lunch\_break\_start} && \wedge \\ &b_{s,d,e,i}.start \leq \textbf{latest\_lunch\_break\_start} && \wedge \\ &b_{s,d,e,i}.length = \textbf{lunch\_break\_length} && \forall s \in S, d \in \{1, ..., D\}, e \in \{1, ..., w_{s,d}\} \end{aligned} \tag{21}$$

Objective

The objective function is the weighted sum of overstaffing understaffing and the number of shifts. In order to define the overstaffing and understaffing we define the *active workers* again. We use the notation $e \in s_d$ to denote the duties which are part of shift s and start on day d.

$$a_t = \sum_{s \in S} x_{s,d}, \text{ where}$$

$$x_{s,d} = \begin{cases} w_{s,d} - \sum_{e \in s_d} y_{e,t}, & \text{if } t \text{ belongs to the interval of the duties of shift } s \text{ that start on day } d \\ 0 & \text{otherwise} \end{cases}$$

$$y_{e,t} = \begin{cases} 1 \text{ if duty } e \text{ is on a break in } t \text{ or } t-1 \\ 0 \text{ otherwise} \end{cases}$$

This means that for each time period we count how many duties are active in that time period (taking duties that are on break in the time period or in the previous time period as not active).

Using the active workers we define the overstaffing and understaffing which lead to the objective function. We will use O to denote the total overstaffing and U to denote the total understaffing. We define them as follows.

$$O = \sum_{t \in T} max\{a_t - R_t, 0\}$$
$$U = \sum_{t \in T} max\{R_t - a_t, 0\}$$

The objective is to minimise the weighted sum of overstaffing, understaffing and the number of shifts.

$$\text{minimize } W_1 \cdot O + W_2 \cdot U + W_3 \cdot |S| \tag{22}$$

3 Literature Review

This section is split up into four different parts. In the first part we will give an overview of literature regarding personnel scheduling. We will also note the differences between most available literature regarding personnel scheduling problems and the shifts and breaks design problem. Afterwards we will give an overview of literature directly related to the shift design problem as discussed in this thesis. Next we will do the same for the break scheduling problem and finally we will give an overview of literature directly related to the shifts and breaks design problem.

3.1 Personnel Scheduling

After the scheduling problem was introduced by Dantzig [11] many different formulations of personnel scheduling problems have been considered. Efficient methods for solving personnel scheduling problems without breaks are available for some specific formulations. The constraint matrix in the integer linear programming formulation of Dantzig [11], who described a set covering problem, is a consecutive ones matrix if the problem contains no cyclicity nor breaks, i.e. all duties are active on a consecutive number of time slots. Matrices with consecutive ones are totally unimodular, as first showed by Veinott and Wagner [32], and hence integer linear programs containing such a constraint matrix can be efficiently solved by solving the linear programming relaxation. Bartholdi et al. [4] show that personnel scheduling problems with a row circular matrix can be solved efficiently by considering at most a bounded number of flow problems. These row circular matrices follow if all duties are of the same length and contain no breaks. Furthermore Bartholdi et al. [4] show that the formulation of Dantzig [11] can be slightly changed to allow for problems in which, next to the linear cost for using a duty, a linear penalty cost for overstaffing and understaffing can be considered for which the efficient solution methods as discussed still hold.

At this point we make a note that, ignoring the problem of minimizing the number of shifts, the constraint matrix for the shift design problem has neither the consecutive ones property, nor the circular row property. The constraint matrix is column circular. Cyclic Scheduling problems involving duties without breaks will always have the circular ones property in the columns. In Hochbaum and Levin [21] the hardness of problems with a column circular matrix is shown to be equivalent to the exact matching problem which is in the complexity class NRC [26]. It is currently unknown whether the problem is in P.

Aykin [2] studied a problem for a continuous 24 hour workday (cyclical) and giving each duty one half hour lunch break and two smaller 15 minute breaks. Each break is allowed to start in a period of 3 to 6 time periods. For this problem an integer linear program formulation is described which differs from original set cover formulation proposed by Dantzig [11] by requiring three breaks to be scheduled. Rekik et al. [30] considered an extension

of the model and tested instances for which exactly three breaks were required for each duty, having a combined duration of two hours. Furthermore, the second break was required to be the longest break and the continuous periods in-between two breaks (working period) was restricted to have a length between one and three hours. For this problem Rekik et al. [30] propose two integer linear program formulations and compare their results with modified versions of integer linear programming formulations given by Bechtold and Jacobs [5] and Aykin [2]. Rekik et al. [30] show that their model is slightly slower than the formulations in [5] and Aykin [2] which is explained by the added flexibility that their model gives. The slight modification considered of the model proposed by Bechtold and Jacobs [5], show that this modified model gives better results than the model proposed by Aykin [2] which is opposed of findings by Aykin [3] who compared the unmodified models.

A recent overview of personnel scheduling is given by Van den Bergh et al. [31]. The authors note that a wide range of solution methods are used to find solution to personnel scheduling problems. The two main approaches used are mathematical programming methods such as linear programming, goal programming, integer programming and column generation, and heuristic methods such as simulated annealing, tabu search and genetic algorithms. Other approaches include simulation, constraint programming and queueing.

The shifts and breaks design problem is different from problems discussed in the literature. In this problem shifts have to be designed, instead of duties, in such a way that the shifts can be efficiently reused over multiple days instead of just a single day. Furthermore the formulation allows for a large number of breaks in the shifts (up to 10 breaks for the real-life instances). An overview of literature regarding shift design and break scheduling is given by Di Gaspero et al. [16].

3.2 Shift Design

The shift design problem was first introduced by [28]. The authors make use of local search to find a solution. Tabu search [19] is used to guide the local search. The authors state that they rely on local search techniques because the shift design problem is proven to be NP hard [23], as well that there exists a constant $c < 1$ such that approximating the shifts design problem within $c \ln n$ is NP hard (here n refers to the total number of time periods in shift design). The authors also use domain knowledge during the search by identifying shifts during the longest period of shortage/excess and trying to improve these shifts. Furthermore a 'good' initial solution is constructed. The main idea for this good solution is that time points in which the demand drastically rises (falls) needs to be a time point for shift starts (ends).

Di Gaspero et al. [14] use a hybrid heuristic, which consists of a greedy heuristic followed by a local search algorithm, which outperforms the previous results. The greedy

heuristic is based on the relation of the shift design problem to a flow problem as described in [23]. A polynomial min cost max flow problem can be used to compute optimal staffing with weighted cost on the overstaffing and understaffing. This approach does however not minimize the number of shifts used. Furthermore this approach can only be used when cyclicity is not considered in the problem. The authors still consider a cyclic problem and do so by making different calls to the greedy heuristic where in each call only a subset of the shifts is considered so that the problem is not cyclic. For the local search algorithm the authors consider less neighbourhood relations as compared to [28]. It should be noted that the problem described in this paper is slightly different to the one described in [28]. Namely in the (chronologically) first article the objective function covers an additional term: a penalty for the *distance of the average number of duties per week*. This term is meant to penalise a set of shifts in which the workload is divided into many small shifts. In the later article the authors note this difference and state that, "In practice further optimization criteria clutter the problem. [...] Fortunately, this and most further criteria can easily be handled by straightforward extensions of the heuristics described in this paper."

Bonutti et al. [7] formulate a variation on the shift design problem. Their formulation extends the shift design problem to have multiple types of skills for employees. The requirements at each time period state the required number of employees at each time period for each skill. For each shift the number of workers for each day and each skill have to be specified. Furthermore in this formulation it is possible for shifts to contain a single break. Local search is used for this problem and the neighbourhood relations are an extension on the relations used in Di Gaspero et al. [14]. Furthermore simulated annealing [22] is used to guide the search process.

Answer Set Programming (ASP) [8] is an exact solution technique and has been applied to shift design by Brewka et al. [8]. ASP is able to find most best known solutions within 60 minutes on instances of shift design where those solutions have no overstaffing and understaffing. While the execution times are often not competitive with results from [14] there are instances on which ASP works very well. Solutions found for the instances in which a solution without overstaffing and understaffing might not exist are not competitive with results from the literature. The authors state that a combination of ASP together with domain-specific heuristics is a promising area for further research.

3.3 Break Scheduling

Break scheduling, as we will be discussing in this thesis, was first introduced by Beer et al. [6]. In their formulation different constraints are given with different weights to each type of violation and there are no hard constraints. Their formulation uses either a random assignment of breaks as an initial solution or a solution following from a simple temporal problem [13]. After the initial solution has been constructed the authors use local search

guided by either tabu search, simulated-annealing, or a minimum conflicts-based heuristic [24]. The best results are obtained by using a minimum conflicts-based heuristic. Beer et al. [6] conclude that the two different initial solutions did not have a significant impact on the solution quality.

A memetic algorithm for break scheduling was proposed by Musliu et al. [29]. Memetic algorithms combine population based method with local search techniques and were first mentioned in [25]. The memetic algorithm keeps a number of individuals, the population, each representing a solution to the break scheduling problem. An individual is made up of memes, in this formulation each meme representing a duty. At each iteration of the algorithm the best individual is kept unaltered for the next iteration. A subset of the other individuals are chosen and either a crossover or mutation operator is applied to each individual in this subset. The crossover operator takes as input two individuals (the second being random) and outputs an individual (offspring) which inherits a part of the solution (memes) of each input (parent). The mutation operator performs a random move on an individual chosen using a local search approach. After this local search is used to improve the m fittest individuals at each iteration. The authors compare their results with the minimum conflicts-based heuristic as proposed by Beer et al. [6]. The algorithms both score best on half of the test instances and hence conclusions regarding the better algorithm are indecisive.

An improved memetic algorithm is given in [33]. In this approach a part of the constraints in the break scheduling problem are considered hard and, ignoring the other constraints which regard the staffing requirements, a simple temporal problem [13] can be formulated. Solutions from this are taken as initial solutions for the memetic algorithm. In this approach memes are defined by a set of time slots and each duty is assigned to exactly one meme in which most of the timeslots of the duty are. The reason for this approach as opposed to the approach used in [29] is that duties have a strong interference is satisfying the staffing requirements, and therefore it is difficult to find effective crossover operations when a meme represents a duty. The authors use a penalty value for each meme based on the last iteration in which the break pattern of that meme was updated. This ensures that memes stuck in local optima are not passed on to offspring. The algorithm also keeps an individual which contains all memes having the best value over all individuals. This ensures that global optima are not discarded. Individuals are also mutated using local search. The algorithm is used on publicly available instances and it finds new best solutions on 28 out of 30 instances. This memetic algorithm is also shown in [34]. In this paper it is also proven that break scheduling is NP-Complete, under the condition that all feasible break patterns of each duty are given explicitly in the input.

3.4 Shifts and Breaks Design

The shifts and breaks design problem is introduced in [15]. The authors state that to the best of their knowledge the whole problem as described in their paper has not been addressed in the literature. Various similar problems have been discussed in the literate as described in section 3.1. The authors propose an approach combining local search (LS) and constraint programming (CP) [1]. Their approach starts of with finding an initial randomly generated solution for the assignment of shifts. LS is employed on a part of the solution, namely a solution which specifies the shifts, the number of duties for each shift on each day, and the number of breaks for each duty. The CP model uses this information to try to find a break allocation within a time limit. For each iteration of the algorithm a random move is chosen which changes either the start or end of a shift by one time unit, the number of duties of a single day for a shift by one duty, the number of breaks for all duties on a single day of a shift, or by merging two shifts together. This new solution is given to the CP model and after the break allocation is determined this new solution is accepted if the objective value is better than the objective value before the random move. The algorithm fixes a part of the break variables in the CP model when a solution to the full program cannot be found in a timely manner.

Since the authors are the first to tackle the shifts and breaks design problem there are no results to compare to. The authors use a set of randomly generated instances as well as a set of real-life instances which are publicly available [12]. The authors used a time limit of one hour for each instance.

4 Shift Design

In this section we will first give an integer linear program (ILP) formulation for the shift design problem. Afterwards we will describe the set of problem instances used to test the effectiveness of this ILP formulation for the shift design problem. Finally we will use the commercially available optimisation package Cplex [10] to solve instances available in the literature and compare our results to results obtained by Di Gaspero et al. [14]. A modified version of the ILP for shift design will also be used in the first phase of our solution approach to the Shifts and Breaks Design problem.

4.1 ILP Shift Design

In this section we give an integer linear programming formulation to the shifts design problem. A compact overview of this ILP is given in the appendix (9.1). We use this ILP to find results to the shift design problem in Section 4.3. A modified version of this ILP is also used in the first phase of our algorithm to the shifts and breaks design problem.

The shift design problem is made up of time periods, days and shifts. We will use the following variables to refer to a single element of each set.

- t refers to a single time period.
- d refers to a single day.
- s refers to a single shift.

The time periods and the days are part of the input, but the shifts are not explicitly given. However we can use the shift types to determine all possible shifts. For each possible starting time of a shift type, all possible lengths are possible. As an example, Table 4 shows all possible shifts for the shift type with min_start=07:00, max_start=08:00, min_length=08:00, max_length=09:00 and a time period of 30 minutes. It should be noted that decreasing the time granularity drastically increases the number of possible shifts. A time granularity of 15 minutes for this example would allow for 25 possible shifts and a time period of 5 minutes allows for 169 possible shifts.

Table 4: Possible Shifts

Start	End
07:00	15:00
07:00	15:30
07:00	16:00
07:30	15:30
07:30	16:00
07:30	16:30
08:00	16:00
08:00	16:30
08:00	17:00

For each shift we are making the decision of using it or not. In the case that a shift is used, we call it active and we can assign a number of duties to it on each day. For these decisions we introduce the following variables.

$$a_s = \begin{cases} 1 & \text{if shift } s \text{ is active} \\ 0 & \text{otherwise} \end{cases}$$

$$w_{d,s} = \text{ the number of duties of shift } s \text{ starting on day } d$$

If a shift is inactive the number of duties on each day are forced to be 0. To model this constraint we use the parameter $M = max_t\{R_t\}$. Recall that R_t denotes the staffing requirements at time period t. Note that using more duties than this number will lead to unnecessary overstaffing. The following constraints force the number of duties to be 0 in case the corresponding shift is not active.

$$w_{d,s} \leq M \cdot a_s \quad \forall d, s \tag{23}$$

Using the number of duties for each shift we can can calculate the overstaffing and the understaffing. For this we need a parameter denoting on which time periods the duties of a shift starting on a specific day are active. We introduce the following.

$$o_t = \text{ the amount of overstaffing at time period } t, \quad o_t \geq 0$$

$$u_t = \text{ the amount of understaffing at time period } t, \quad u_t \geq 0$$

$$A_{s,d,t} = \begin{cases} 1 \text{ if the duties of shift } s \text{ starting on day } d \text{ are active on time period } t \\ 0 \text{ otherwise} \end{cases}$$

The following constraints specify, respectively, the understaffing and the overstaffing.

$$\sum_{s,d} \left(A_{s,d,t} \cdot w_{d,s}\right) + u_t \geq R_t \qquad \forall t \tag{24}$$

$$o_t = \sum_{s,d} \left(A_{s,d,t} \cdot w_{d,s} \right) + u_t - R_t \qquad \forall t \tag{25}$$

In an optimal solution if $u_t > 0$ then (24) will be an equality. To see this note that if the left hand side would be strict then we could decrease u_t which would decrease the objective. The objective function is the weighted sum of the overstaffing, understaffing and the number of shifts.

Results of this ILP on instances of shift design are given in Section 4.3. In our two-phase approach to the Shifts and Breaks design problem we use a modified version of this ILP. The modified ILP is based on using virtual shifts which will be described in the next section.

$$\text{minimize } W_1 \sum_t u_t + W_2 \sum_t o_t + W_3 \sum_s a_s \tag{26}$$

4.2 Problem Instances

Instances used for testing the integer linear programming approach for shift design are available online at [20]. We compare the results of our integer linear programming approach to results obtained by Di Gaspero et al. [14]. Hereto we use the first 30 instances from the first set and all 30 instances of the third set of the publicly available instances. The instances are created by randomly selecting a set of shifts and number of duties per shift for each day. The time granularity for these instances is randomly chosen to be either 15, 30, or 60 minutes. The weights for these instances were $W_1 = W_2 = 1$. W_3 was chosen to be equal to the length of a time period. Instances of the first set only feature legal shifts and therefore the existence of a solution in which there is no overstaffing and understaffing is guaranteed. The instances of the third set are constructed using shifts for which the starting time and/or length can exceed that of the given shift types. Therefore the existence of a solution providing exact coverage to the requirements is not guaranteed.

These instances allow for a total of 39 possible shifts for instances with a time granularity of 60 minutes, 110 for instances with a time granularity of 30 minutes and 360 for instances with a time granularity of 15 minutes. The instances contain an average of 10 shifts with an average of 3 duties per day.

Table 5: Shift Types

Abbr.	Name	Earliest start	Latest start	Minimum length	Maximum length
M	Morning Shift	05:00	08:00	7:00	9:00
D	Day Shift	09:00	11:00	7:00	9:00
E	Evening Shift	13:00	15:00	7:00	9:00
N	Night Shift	21:00	23:00	7:00	9:00

4.3 Results

The integer linear program for shift design was written in C++ using Microsoft Visual Studio 2013 [9], using the commercially available optimisation package Cplex [10] to solve it.

In this section we compare the results of our ILP approach to the results obtained by Di Gaspero et al. [14]. Overall the authors found their best results using a combination of a greedy min-cost max-flow formulation and local search. The authors refer to this approach by 'GrMCMF+LS'. We compare our results to the results of GrMCMF+LS.

For the instances of the first data set the authors were interested in the time it took for their approach to reach the 'best known' solutions of shift design. The best known solutions are equal to either the solution used for constructing the instance or a better solution as found in their article. The authors report average running time of their algorithm to obtain the best known solutions. For one of the instances GrMCMF+LS was unable to find the best known solution.

Using integer linear programming we were able to find optimal solutions to all instances of the first set allowing a time limit of up to 30 minutes. We show our results in Table 6. For each instance we report the 'best known' solution and the optimal solution found by the integer linear program. We have highlighted the instances were the optimal solution was lower than the best known solution. Furthermore we show the average running time of GrMCMF+LS and the running time of the ILP rounded to the nearest second.

Di Gaspero et al. [14] used the third dataset to determine the quality of quickly found solutions. Hereto the authors ran each instance 100 times using a time limit of 1 second and reported the average solution value. For our comparison we ran the ILP model two times, the first time also with a 1 second time limit and the second time with a 30 minute time limit. Results are shown in Table 7. For the ILP with a 1 second time limit we report

the found solution and *gap* of this solution. In Cplex the gap is defined as

$$\frac{|\text{bestnode} - \text{bestinteger}|}{10^{-10} + |\text{bestinteger}|} \tag{27}$$

where *bestnode* is a lowerbound to the problem, a solution of a linear relaxation of the ILP, and *bestinteger* is the current best solution to the ILP. Thus the gap is the ratio of a difference between the best found solution and a lowerbound divided by the lowerbound. For the ILP with a time limit of 30 minutes we also report the found solution and the gap, indicated by 'ILP+'. We also report the running time of the ILP in seconds, which can be lower than 1800 if the optimal solution is found.

The results show that ILP obtains better results than the average value obtained by GrM-CMF+LS. Furthermore the ILP is able to find and proof an optimal solution in 54 out of the 60 instances. All the instances for which optimality could not be proved are part of the third set. This shows evidence for the idea that the instances constructed from a feasible seed solution might be biased since a solution providing exact coverage need not to exist.

Table 6: Shift Design Comparison Set 1

Instance	Objective		Time (secs)	
	GrMCMF+LS	ILP	GrMCMF+LS	ILP
1-1	480	**320**	1	2
1-2	300	**212**	40	21
1-3	600	**374**	2	2
1-4	450	**340**	109	165
1-5	480	**319**	2	2
1-6	420	**213**	1	1
1-7	270	**237**	7	1
1-8	150	**147**	11	149
1-9	150	**149**	9	29
1-10	330	**289**	84	141
1-11	30	30	1	1
1-12	90	**81**	4	3
1-13	105	105	4	9
1-14	195	**187**	61	425
1-15	180	**170**	0	1
1-16	225	**209**	152	1552
1-17	540	**394**	288	283
1-18	720	**447**	7	6
1-19	180	**177**	31	54
1-20	540	**353**	2	2
1-21	120	**119**	2	7
1-22	75	75	4	4
1-23	150	150	22	100
1-24	480	**343**	1	6
1-25	480	**352**	n.a.	168
1-26	600	**347**	9	5
1-27	480	**393**	2	3
1-28	270	**222**	4	32
1-29	360	**289**	10	58
1-30	75	75	2	2
Average	318	237	30	108

Table 7: Shift Design Comparison Set 3

Instance	Objective			Gap		Time (secs)
	GrMCMF+LS	ILP	ILP+	ILP	ILP+	ILP+
3-1	2386	318	318	0.31	0	8
3-2	7691	845	536	0.58	0	87
3-3	9597	924	542	0.55	0	25
3-4	6681	1427	500	0.77	0	119
3-5	9996	551	508	0.28	0	5
3-6	2077	1892	315	0.89	0	65
3-7	6087	642	585	0.28	0	72
3-8	8861	725	537	0.45	0	47
3-9	6036	2527	570	0.83	0	420
3-10	3002	462	366	0.52	0	62
3-11	5491	1024	474	0.73	0	195
3-12	4171	3514	520	0.91	0.04	1800
3-13	4662	3131	535	0.89	0.07	1800
3-14	9661	701	461	0.42	0	3
3-15	11445	1112	641	0.57	0	151
3-16	10734	638	477	0.38	0	3
3-17	4729	3011	523	0.89	0.04	1800
3-18	6692	893	526	0.67	0	200
3-19	5157	2677	594	1	0.13	1800
3-20	9175	1845	606	0.80	0	232
3-21	6054	4674	718	1	0.10	1800
3-22	12870	2063	720	0.76	0.03	1800
3-23	8390	699	630	0.42	0	18
3-24	10418	741	590	0.43	0	5
3-25	13252	847	635	0.41	0	5
3-26	13118	1042	724	0.50	0	193
3-27	10081	1034	633	0.56	0	7
3-28	10604	887	539	0.53	0	5
3-29	6690	1045	571	0.69	0	263
3-30	13724	1011	607	0.53	0	6
Average	7984	1430	550	0.62	0.01	433

5 Break Scheduling

In this section will formulate a solution approach to the break scheduling problem. We will first formulate the problem of assigning an optimal break allocation for a single duty as an integer linear program. Afterwards we state how this problem can be modified so that multiple duties can be considered at the same time. We will propose two algorithms which use these ILP models to find break patterns for all duties. We will compare results of both our algorithms as well as results obtained by Widl and Musliu [33].

5.1 ILP Break Scheduling Single Duty

In this section will formulate the problem of optimally allocating the break allocation for a single duty. A compact overview of this ILP is given in the appendix 9.2.

First we will define the set of possible breaks for a duty. Since the total break time for a duty is given, as well as the minimum break length, we can calculate the maximum possible number of breaks in a duty. We let M_{breaks} be this maximum number of breaks+1. The 1 is added such that we can use the same set to denote the working periods, since there is always 1 more working period than breaks. We use the following set to enumerate all breaks.

$$B = \{1, 2, ..., M_{breaks}\} \tag{28}$$

For each (possible) break we first need to decide whether the break will be active, and in case a break is active we need to decide the starting period and the length of the break. We use the binary variables a_b to indicate whether break b is active. First we impose the following constraint which force the breaks to be in chronological order since a break can only be active if the previous break was also active.

$$a_b \leq a_{b-1} \quad \forall b \in B \setminus \{1\} \tag{29}$$

Breaks following a long working period are considered different since they have a different minimum length. Furthermore, one of the breaks (might) need to be assigned the status of lunch break, which requires it to be somewhere in the middle of the duty and have a specific length. We will use the binary variables a_b^l to indicate whether break b needs to be long, and the binary variables l_b to denote whether break b is assigned the status of lunch break. Only active breaks can be long or lunch and therefore we require the following constraint.

$$a_b^l + l_b \leq a_b \quad \forall b \in B \tag{30}$$

This constraint also ensures that an (active) break can be either long or lunch but not both.

We use integer variables b_b^l to denote the length of the bth break in number of time periods. Since the total break time ($Tbt =$ **total_break_time**) is given we require the following constraint.

$$\sum_b b_b^l = Tbt \tag{31}$$

In order to enforce some of the logical constraints we will use M to denote a sufficiently large variable, in this case M will be equal to the length of the duty expressed in number of time periods. First we set a minimum and maximum length on the breaks, using $Bminl =$ **break_minimum_length** and $Bmaxl =$ **break_maximum_length**, on all active break lengths. Note that the length of inactive breaks gets forced to 0. The lunch break length might exceed the maximum normal break length hence we also account for this by making the constraint inactive for the break assigned the status of lunch break.

$$M \cdot (1 - a_b) + b_b^l \geq Bminl \qquad \forall b \in B \tag{32}$$

$$b_b^l \leq Bmaxl \cdot a_b + M \cdot l_b \qquad \forall b \in B \tag{33}$$

Long breaks will have a different minimum length. This is specified by the following constraint using $Lbml =$ **long_break_minimum_length**.

$$a_b^l \cdot Lbml \leq b_b^l \quad \forall b \in B \tag{34}$$

The variables b_b^s will correspond to the start of break b i.e. the first time period on which the respective break is active. We use t to refer to the time periods in this ILP. Later we will introduce constraints such that the break start is forced to be on a time period. Using that the duty lasts a total of $Length$ time periods, the set of time periods is given by

$$T = \{1, 2, ..., Length\} \tag{35}$$

Breaks can not start too close to the shift extremes ($Ebs =$ **earliest_break_start** and $Lbs =$ **latest_break_ start**). Therefore we add the following constraints which are only active for active breaks.

$$b_b^s \geq Ebs \qquad \forall b \in B \tag{36}$$

$$b_b^s \leq Length - Lbs + (1 - a_b) \cdot (Lbs + 1) \qquad \forall b \in B \tag{37}$$

Constraint (37) combined with Constraint 38 ensures that the break start of inactive breaks is set to the first time period following the shift end. This is useful for defining the working periods.

$$b_b^s \geq (Length + 1) \cdot (1 - a_b) \quad \forall b \in B \tag{38}$$

We let the variables wp_b denote the length (in time periods) of the bth working period. The first working period starts at the first time period of the duty and ends one period

before the first break starts. For the other working periods, the bth working period start after the $b-1$th break has ended and ends when break b starts.

$$wp_1 = b^s_1 - 1 \tag{39}$$
$$wp_b = b^s_b - (b^s_{b-1} + b^l_{b-1}) \qquad \forall b \in B \setminus \{1\} \tag{40}$$

Active working periods have a lower and upper bound on their length. These are given by the parameters $Wpminl =$ **working_period_minimum_length** and $Wpmaxl =$ **working_period_maximum_length**. Note that a working period wp_b is active if break $b-1$ is active, and that wp_1 is always active.

$$wp_b \leq Wpmaxl \qquad \forall b \in B \tag{41}$$
$$wp_1 \geq Wpminl \tag{42}$$
$$M \cdot (1 - a_{b-1}) + wp_b \geq Wpminl \qquad \forall b \in B \setminus \{1\} \tag{43}$$

The following constraints ensure, respectively, that a^l_b or $l_b$is forced to 1 if break b follows a long working period and to 0 otherwise. Hereto we use the parameter $Mlwp =$ **minimum_long_working_period**. We have earlier specified that a break can be either long or lunch, and only if it is active, and therefore we let the first constraint be inactive for the break which is the lunch break. By doing this we make the assumption that **minimum_long_working_period** $\leq$ **lunch_break_length**. The constraint is also inactive for inactive breaks, this is necessary because otherwise we could force an inactive break to be long which is not feasible.

$$wp_b - Mlwp + 1 \leq a^l_b \cdot M + l_b \cdot M + (1 - a_b) \cdot M \qquad \forall b \in B \tag{44}$$
$$Mlwp - wp_b \leq (1 - a^l_b) \cdot M \qquad \forall b \in B \tag{45}$$

If a lunch break is required there needs to be exactly one lunch break. We will use the parameter L which will be equal to 1 if a shift needs a lunch break and 0 otherwise. We have already introduced l_b, indicating whether a break is the lunch break. Therefore we require the following.

$$\sum_b l_b = L \tag{46}$$

A lunch break has a fixed length of $Lbl =$ **Lunch_break_length**, a minimum start at $Elbs =$ **Earliest_lunch_break_start**, and a maximum start at $Llbs =$ **Latest_lunch_break_start**. In order to ensure that the break which will be the lunch break does not violate any of these constraints we add the following four types of constraints. Respectively the constraints ensure that the lunch break does not: last shorter than allowed, last longer than allowed, start earlier than allowed, and that the lunch break does not start later than allowed.

$$(1 - l_b) \cdot M \geq Lbl - b^l_b \qquad \forall b \in B \tag{47}$$

$$(1 - l_b) \cdot M \geq b_b^l - Lbl \qquad \forall b \in B \qquad (48)$$
$$(1 - l_b) \cdot M \geq Elbs - b_b^s \qquad \forall b \in B \qquad (49)$$
$$(1 - l_b) \cdot M \geq b_b^s - Llbs \qquad \forall b \in B \qquad (50)$$

Up until this point we have defined variables which specify all breaks. What is left is to define variables for the overstaffing and understaffing. To do this we need to specify whether the duty is active at a time period. In order to do this we add binary variables $z^s_{t^*,b}$ and $z^e_{t^*,b}$. Note that we use t^* to index these variables rather than t. This is done to highlight the fact that, since inactive breaks have their break start and break end one time period after the last time period of the duty, an additional time period is needed for these variables. Therefore we use the set T^* for these variables instead of T where

$$T^* = \{1, 2, ..., Length, Length + 1\} \qquad (51)$$

$$z^s_{t^*,b} = \begin{cases} 1 \text{ if time period } t^* \text{ is the first time period on which the } b\text{th break is active} \\ 0 \text{ otherwise} \end{cases}$$

$$z^e_{t^*,b} = \begin{cases} 1 \text{ if time period } t^* \text{ is is the last time period on which the } b\text{th break is active} \\ 0 \text{ otherwise} \end{cases}$$

All other variables of the ILP can be derived from these two types of variables since they specify all breaks. Furthermore the variables $z^s_{t^*,b}, z^e_{t^*,b}, a_b, a^l_b$, and l^b are the only binary variables in the ILP model. The other variables, such as the break start b^s_b and the break length b^l_b can be relaxed to be continuous. We will add constraints to ensure that these variables only take on integer values.

We use the following 4 types of constraints to determine the break start and length using the variables $z^s_{t^*,b}$ and $z^e_{t^*,b}$. Constraint (52) ensures that exactly 1 time period is taken as the break start. Constraint (53) constraint ensures that the continuous variables b^s_b are correctly taken based on $z^s_{t^*,b}$. Constraint (54) ensures that exactly 1 time period is taken as the break end. Constraint (55) ensures that the continuous variables b^l_b are correctly taken based on $z^e_{t^*,b}$, note that the term $1 - a_b$ is needed to correct the constraint for inactive breaks since inactive breaks have a length of 0.

$$\sum_{t^*} z^s_{t^*,b} = 1 \qquad \forall b \in B \qquad (52)$$
$$\sum_{t^*} t^* \cdot z^s_{t^*,b} = b^s_b \qquad \forall b \in B \qquad (53)$$

$$\sum_{t^*} z^e_{t^*,b} = 1 \qquad \forall b \in B \tag{54}$$

$$\sum_{t^*} t^* \cdot z^e_{t^*,b} = b^s_b + b^l_b - 1 + (1 - a_b) \qquad \forall b \in B \tag{55}$$

The following two types of continuous variables are used to determine whether a duty is on a break in a time period. This is necessary for verifying whether a duty is active on a time period. We use the following variables.

$$z^b_{t,b} = \begin{cases} 1 \text{ if time period } t \text{ is before the start of the } b\text{th break} \\ 0 \text{ otherwise} \end{cases}$$

$$z^a_{t,b} = \begin{cases} 1 \text{ if time period } t \text{ is after the } b\text{th break has ended} \\ 0 \text{ otherwise} \end{cases}$$

The following two constraints ensure that these variables take on the correct value. Constraint (56) ensures that $z^b_{t,b}$ is correctly defined, note that a time period is before a break if the break start is after the time period. Constraint (57) ensures that $z^a_{t,b}$ is correctly defined, note that a time period is after a break if the break end is before the time period.

$$\sum_{i=t+1}^{Length+1} z^s_{i,b} = z^b_{t,b} \qquad \forall t \in T, b \in B \tag{56}$$

$$\sum_{i=1}^{t-1} z^e_{i,b} = z^a_{t,b} \qquad \forall t \in T, b \in B \tag{57}$$

We will use the variables $z_{t,b}$ denote whether the duty is on its bth break on time period t. A duty is on its bth break at time period t if the time period is not before the break start and not after the break end. We have already introduced variables to indicate these cases and since a time period is either before a break, during a break, or after a break we use the following constraint to determine $z_{t,b}$.

$$z_{t,b} + z^a_{t,b} + z^b_{t,b} = 1 \quad \forall t \in T, b \in B \tag{58}$$

x_t will indicate whether the duty is active on time period t. This is true if and only if the duty is not on any break in the time period nor on the time period before. The following constraints do the following. Constraint (59) forces the duty to be inactive on a time period if the duty is on a break. Constraint (60) forces the duty to be not active in case a break ended on the previous time period. Constraint (61) forces the first period to be active in case the duty is on no breaks in the first time period. Constraint (62) forces the other time

periods of the duty to be active in case the duty is not on any break and no break ended the previous time period.

$$1 - x_t \geq \sum_b z_{t,b} \qquad \forall t \in T \tag{59}$$

$$1 - x_t \geq \sum_b z^e_{t-1,b} \qquad \forall t \in T \setminus \{1\} \tag{60}$$

$$1 - x_1 \leq \sum_b z_{1,b} \tag{61}$$

$$1 - x_t \leq \sum_b z_{t,b} + \sum_b z^e_{t-1,b} \qquad \forall t \in T \tag{62}$$

The last variables we define are the overstaffing and understaffing variables. These will be used in the objective function. For this ILP we assume that the allocation of other shifts is fixed. This fixed allocation together with the assumption that the duty, for which the breaks are to be decided, takes no breaks, make up the fixed staffing. Let us use D_t to determine the demand at time period t which is a parameter used in this ILP. D_t is equal to the requirements at period t minus the fixed staffing at period t. Note that the duty considered in the ILP is considered to have no breaks while determining the fixed staffing. We use the following two (in)equalities to base the overstaffing and understaffing constraints on.

$$\text{fixed staffing} + x_t - 1 + \text{understaffing} \geq \text{requirements}$$
$$\text{overstaffing} = x_t - 1 + \text{understaffing} + \text{fixed staffing} - \text{requirements}$$

Then the following constraints specify the over and understaffing, using o_t to define the overstaffing at time period t and u_t to define the understaffing at time period t.

$$x_t - 1 + u_t \geq D_t \qquad \forall t \in T \tag{63}$$

$$o_t - u_t - x_t + 1 = -D_t \qquad \forall t \in T \tag{64}$$

$$o_t, u_t \geq 0 \tag{65}$$

Finally we specify the objective function.

$$\text{minimize } W_1 \sum_t o_t + W_2 \sum_t u_t \tag{66}$$

5.2 ILP Break Scheduling Multiple Duties

The ILP of the previous section can easily be extended to allow for multiple duties to be considered at the same time as long as these duties all have the same starting time and length. Therefore it can be used to optimally allocate all duties starting on the same

day of a shift. For this ILP all variables as described in the previous section, except for the understaffing and overstaffing, need an additional subscript e indicating the duty. Then all constraints except the constraints specifying understaffing, Constraint 63, and overstaffing, Constraint 64, need to hold for all duties. These two mentioned constraints should be replaced by the following constraints.

$$\sum_{e} \big(x_{t,e} - 1\big) + u_t \geq D_t \qquad \forall t \in T \tag{67}$$

$$o_{t,e} - u_{t,e} - \sum_{e} \big(x_{t,e} + 1\big) = -D_t \qquad \forall t \in T \tag{68}$$

5.3 Algorithms for Break Scheduling

In this section we will propose two algorithms used to allocate all breaks using the break scheduling ILP's from the previous two sections. The first algorithm will only use the break ILP model from Section 5.1 which allocates the break pattern for a single duty. The second algorithm will also use the ILP from section 5.2 and use it to find the optimal break allocation for two duties at the same time. In Section **??** we compare results of these two algorithms on instances of break scheduling. After introducing the two algorithms we will show a way to improve the running time of both algorithms by not considering the break allocation of duties for which nothing changed since their break allocation was last considered.

5.3.1 Using Single Duties

We will give a short description of the algorithm used to allocate all break allocations using only the ILP model from Section 5.1. Pseudocode of this algorithm is also shown. The algorithm starts with a random duty and assigns a break allocation as specified by the solution of the break scheduling ILP for a single duty. Afterwards the number of active workers at each time period is updated and a break allocation is found for the next duty. After all duties have been considered by this approach the algorithm starts over again with the first duty. This process continuous until all duties have been considered in a row without the objective function of the break scheduling problem changing.

Algorithm 1 shows the pseudocode for this algorithm. We use E to denote the set of all duties. Since E contains all duties we can use it to determine the initial Active_Workers (assuming no breaks). We will use $Obj(E)$ to denote the objective value to the shifts and breaks design problem using the duties in E (and their possible break allocation).

5.3.2 Using Double Duties

Starting with no break allocations we will find an initial break allocation for each duty by considering them by one at a time using the ILP model from Section 5.1. After each

```
Initialise Active_Workers using E (assuming no breaks)
Current_Objective = Obj(E)
Previous_Objective = ∞
while Current_Objective<Previous_Objective do
    for e ∈ E do
        if e has a break allocation then
            remove break allocation from e
            update Active_Workers
        end
        Solve Break Scheduling ILP for duty e using Active_Workers
        Use solution from ILP to allocate breaks to e
        update Active_Workers
    end
    Previous_Objective=Current_Objective
    Current_Objective=Obj(E)
end
```

Algorithm 1: Second Phase Algorithm Using Single Duties

duty has a break allocation we will use the ILP from Section 5.2 to optimally allocate the breaks for two duties at the same time. Hereto we first search for any pair of two duties, having same start time and length, and having at least one time period in common in which there is understaffing and both of the duties are inactive. Since the duties need to have the same starting time and length the duties have to belong to the same shift and day. The reasoning for looking for duties satisfying these properties is that time periods with understaffing are where most of the improvements can be found since the penalty for understaffing is higher in our instances than the penalty for overstaffing. We keep doing this until there is no such pair of duties, or until all shifts and days have been considered in a row without finding improvements.

Pseudocode for this algorithm is shown in Algorithm 2. We will use S to denote the set of all shifts and D to denote the set of all days. In the pseudocode we use the following notation.

$T(s,d)$ Time periods belonging to the duties of shift s starting on day d

$E(s,d)$ Duties of shift s starting on day d

$$e(t) = \begin{cases} 1 \text{ if duty } e \text{ is active on time period } t \\ 0 \text{ if duty } e \text{ is not active on time period } t \end{cases}$$

```
Initialise Active_Workers using E (assuming no breaks)
for e ∈ E do
    Solve Break Scheduling ILP for duty e using Active_Workers
    Use solution from ILP to allocate breaks to e
    update Active_Workers
end
Current_Objective = Obj(E)
Previous_Objective = ∞
while Current_Objective<Previous_Objective do
    for s ∈ S do
        for d ∈ D do
            for e ∈ E_{s,d} do
                for e' ∈ E_{s,d} \ {e} do
                    if t ∈ T_{s,d} where e(t) + e'(t) = 0 then
                        remove break allocation from e and e'
                        Solve Break Scheduling ILP Multiple duties for e and e'
                        Use solution from IP to allocate breaks of e and e'
                        update Active_Workers
                    end
                end
            end
        end
    end
    Previous_Objective=Current_Objective
    Current_Objective=Obj(E)
end
```

Algorithm 2: Second Phase Algorithm Using Two Duties

5.3.3 Improvement

Both algorithms naively loop over all duties. However each duty only influences a subset of the time periods on the planning horizon. During the first round of the while loop all duties should be considered as they all obtain a new break allocation. However it might be possible that there is a single duty which has no time periods overlapping with another duty. It would be useless to consider this duty again in the next while loop since the fixed staffing has not changed and hence the break scheduling ILP will return the same solution. It is also possible that during the third round of the while loop some duties are considered for the break ILP but the fixed staffing of the time periods of that duty have not changed to the situation in the second while loop. In this case we also do not have to solve the break allocation ILP since we already know the optimal solution.

In order to check whether the fixed staffing for the break scheduling ILP of a duty has changed we will keep track of the variables $F_{t,e}$ which we define as follows.

$$F_{t,e} = \begin{cases} 1 \text{ if time period } t \text{ is part of duty } e \text{ and the break allocation has changed last iteration} \\ 0 \text{ otherwise} \end{cases}$$

We use these variables to check whether we will solve a break scheduling problem for a duty or not. We will solve a break scheduling problem for duty e if there exists at least one other duty which has at least one time period in common with duty e and for which the break allocation changed in the last iteration. By abuse of notation we will write $t \in e$ to denote that time period t is part of duty e. Using these variables we can rewrite Algorithm 1 to Algorithm 3 which is shown on the next page.

5.4 Problem Instances

For testing our algorithms for break scheduling we used instances available at [12]. We test our approach to compare results to approaches used in the literature. In total we test our algorithm on 10 different instances. The instances are based on randomly generated instances for shift design. For each of the constructed solutions containing no overstaffing and understaffing, a feasible break allocation was computed for each duty. The number of active workers at each time period using this break allocation is taken to be the staffing requirement for that time period. The constructed shifts without breaks are given. For these randomly generated instances it is therefore guaranteed that a break allocation for each duty can be found such that the staffing requirement are met exactly.

Initialise $Active\_Workers$ using E (assuming no breaks)
$Current\_Objective = Obj(E)$.
$Previous\_Objective = \infty$;
$F_{t,e} = 1 \quad \forall t, e$
while *Current_Objective*<*Previous_Objective* **do**
  for $e \in E$ **do**
    if $\exists \quad e' \neq e$ *s.t.* $F_{t,e'} = 1$ ***and*** $t \in e$ **then**
      if e *has a break allocation* **then**
        remove break allocation from e
        update $Active\_Workers$
      end
      Solve Break Scheduling ILP for duty e using $Active\_Workers$
      Use solution from ILP to allocate breaks to e
      Update $Active\_Workers$
      Update $F_{t,e} \quad \forall t$.
    end
  end
  $Previous\_Objective$=$Current\_Objective$
  $Current\_Objective$=$Obj(E)$.
end

Algorithm 3: Second Phase Updated Algorithm Using Single Duties

Table 8: Parameters

Parameter	**Value**
Days	7
Time Granularity	5 minutes
W_1	2
W_2	10
total_break_time (L = length of duty in minutes)	$\begin{cases} \text{if } L \leq 10 \text{ hours} & \lfloor (L-20) \div 50 \rfloor \cdot 10 \\ \text{else} & L \div 4 \end{cases}$
break_minimum_length	10 minutes
break_maximum_length	60 minutes
working_period_minimum_length	30 minutes
working_period_maximum_length	100 minutes
long_working_period	50 minutes
long_break_minimum_length	20 minutes
earliest_break_start	30 minutes (from duty start)
latest_break_start	30 minutes (from duty end)
min_length_for_lunch	360 minutes (6 hours)
earliest_lunch_break_start	210 minutes (from duty start)
latest_lunch_break_start	330 minutes (from duty start)
lunch_break_length	30 minutes

The parameters defining these instances are given in 8. For the break scheduling ILP, these parameters allow for a total of $336,599$ possible break allocations for duties with a length of 7 hours and duties with a length of 8 hours have $1,878,678$ possible break allocations. These numbers were found by initialising a break scheduling ILP to Cplex with an empty objective value. Cplex offers a *populate* method which can be used to find all optimal solutions. Since the objective function is empty all feasible solutions are optimal. We also tried this method to find the number of feasible break allocations for duties of length 9. After 30 minutes Cplex found over $6,000,000$ possible solutions, however at this point we interrupted the search so more solutions might be possible.

A solution to the break scheduling problem corresponds to exactly one break allocation. This can be verified by noting that the variables a_b, l_b, b_b^l, and b_b^s uniquely define a solution. These variables define all other variables. The only possible exception to this would be if the solution could change the lunch break status from one break to another. However changing the constrains such that at least two lunch breaks were necessary yielded 0 solutions for all duty lengths and hence we can exclude this possibility.

Table 9: Results Break Scheduling

Instance	Overstaffing		Understaffing		Objective			Time (mins)	
	Single	Double	Single	Double	Memetic	Single	Double	Single	Double
1-1	152	268	61	188	**440**	914	2416	20	30
1-2	185	369	109	303	**476**	1460	3768	15	30
1-5	157	306	85	239	**418**	1164	3002	15	30
1-7	179	306	73	230	**583**	1088	2912	21	30
1-9	199	339	122	263	**423**	1618	3308	15	30
1-13	129	227	77	186	**445**	1028	2314	15	30
1-24	190	274	110	213	**611**	1480	2678	15	30
1-28	139	219	91	178	**318**	1188	2218	11	30
2-1	233	361	121	261	**889**	1676	3332	24	30
2-4	188	334	109	265	**535**	1466	3318	16	30
Average	175	300	96	232	**514**	1308	2927	17	30

5.5 Results

The algorithm for breaks scheduling was written in C++ using Microsoft Visual Studio 2013 [9]. The integer linear programs were solved using the commercially available optimisation package Cplex [10].

For break scheduling we compare algorithms 1 and 2 which we refer to by 'Single', and 'Double', respectively. Recall that the 'Single' algorithm only considers the break allocation for a single duty at a time and the 'Double' algorithm also considers the break allocation for two duties. For both algorithms we also used to variables $F_{t,e}$ to skip unnecessary break scheduling ILP's. In Table 9 results are shown. We also show results by Widl and Musliu [34]. For our approach we report the number of overstaffing and understaffing. The objective function for all three approaches is equal to the value in break scheduling and hence there is no penalty for the number of shifts used. For our approaches we also show the running time in minutes, we allowed a maximum of 30 minutes.

Widl and Musliu [34] used a memetic algorithm and gave the instances a running time of 50 minutes. The reported values for the memetic algorithm are average results over 10 runs per instance. Note that the memetic algorithm outperforms our approaches and algorithm 1 outperforms 2. Our first algorithm however converges to a local minimum since the running time of 30 minutes is never reached. Our second algorithm can not consider as much ILP models as the first algorithm since optimally allocating the breaks for two duties at the same time takes much longer. Generally the most time taken for the break allocation of a single duty was around 10 seconds while for two duties at the same time the allocation took multiple minutes.

6 Shifts and Breaks Design

In this section we will first describe our approach used for finding solutions to the shifts and breaks design problem. Hereto we first introduce *virtual shifts*. These virtual shifts are used to modify the shift design ILP introduced in 4.1. This modified ILP will form the first phase of our solution approach. The second phase will use Algorithm 3 to allocate breaks to all duties. This algorithm uses only the break scheduling ILP for a single duty.

We decided to use a two-phase approach since the search space of the shifts and breaks design problem is enormous (2800 possible shifts with 7 days leading to 19600 possible different duties and over $1,000,000$ possible break allocations per duty). We choose to make use of integer linear programming in each phase. Integer linear programming is a widely used method for personnel scheduling problems. The literature review of Van den Bergh et al. [31] shows that it is the most used method. An advantage of the approach is that many problems are easily formulated as an ILP and good solvers for ILP are available. Furthermore ILP is an exact method and will therefore return an optimal solution if time permits. If this can not be done in a timely manner then the best found solution, as well as a lowerbound to the problem are still given by ILP solvers.

After having shown our solution method we describe the set of randomly generated instances used to test it. We will then compare our results to results obtained by Di Gaspero et al. [15]. Based on these results we will propose an improvement for the first phase of our algorithm. We will show results for this improvement on the randomly generated instances as well as 5 real life instances used by Di Gaspero et al. [15].

6.1 Virtual Shifts

While finding a set of shifts in the first phase we want to account for the break time which will be allocated in the second phase. In order to do this we introduce the concept of *virtual shifts*. We will associate a virtual shift for each possible shift. A shift is either active in a time period or it is not. We allow virtual shifts to be only partially active. In the shift design ILP this means that the parameters $A_{s,d,t}$, which were binary and indicated whether duties of shift s starting on day d were active on time period t, can take any value between 0 and 1.

In order to define values for $A_{s,d,t}$ we use the number of inactive time slots for a shift compared to the number of active time slots. A duty is not active on their breaks and on the first time period following a break. The total break time required for a duty is given as part of the input by the parameter **total_break_time**. Before assigning the breaks we

do not know the number of breaks that a duty will have and hence we can not determine the number of inactive periods of that duty exactly. In order to calculate the number of inactive periods, let us denote this by IP, we will assume breaks of minimum length. This assumption is done since it gives the most inactive time periods. Since the penalty for overstaffing is smaller than the penalty for understaffing, at least in the tested instances, we rather aim for too much staffing than too little. Furthermore we take into account the possible need for a lunch break while calculating IP. In short: IP is made up of 3 parts, the **total_break_time**, the inactive period following each break of minimum length, and possible the inactive period following the lunch break. We determine IP by the following equation assuming that all parameters are given in number of time periods. We use L which will be equal to 1 if the shift requires a lunch break.

$$IP = \textbf{total\_break\_time} + \left\lceil \frac{\textbf{total\_break\_time} - L \cdot \textbf{lunch\_break\_length}}{\textbf{break\_minimum\_length}} \right\rceil + L \quad (69)$$

For each duty we know that the first **earliest_break_start** time periods and the last **working_periods_minimum_length**-1 time periods can not contain any breaks. In the input we considered there were a lot of feasible break allocations. There did not seem to be a further bias for certain time slots having a higher chance probability of being inactive. Therefore we 'divide' the rest of the inactive periods evenly over the remaining periods of the shift. Let us use R_s^* to denote the ratio of inactive periods to active periods of shift s, excluding the first **earliest_break_start** and last **working_periods_minimum_length**-1 time periods.

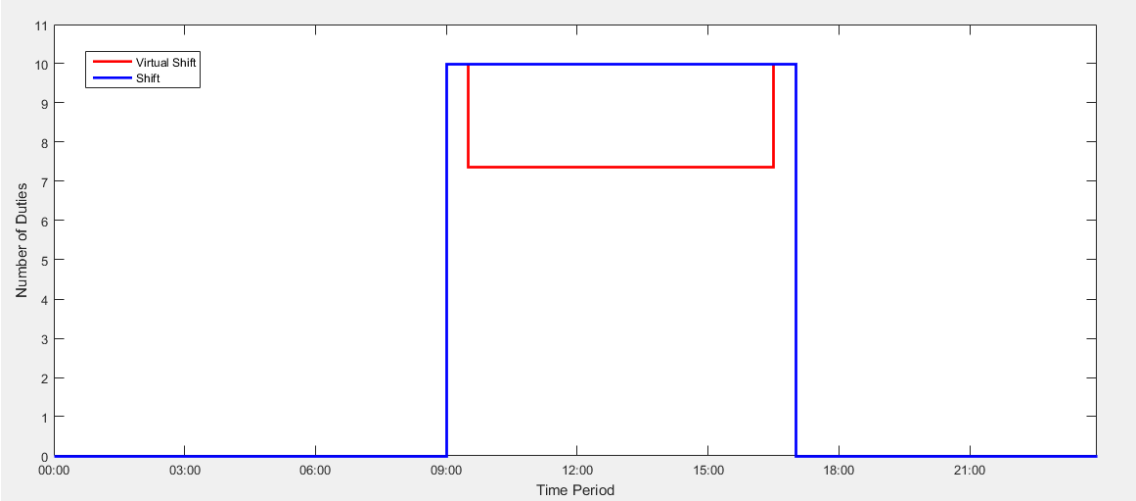
$$R_s^* = \frac{IP}{s.length - \textbf{earliest\_break\_start} - (\textbf{working\_period\_minimum\_length} - 1)} \quad (70)$$

Using IP_s to denote the number of inactive periods for shift s we define the parameters $A_{s,d,t}$ for the virtual shifts as follows, using ebs = **earliest_break_start** and $wpml$ = **working_period_maximum_length**.

$$A_{s,d,t} = \begin{cases} 0 \text{ if } t \text{ is not part of the duties of shift } s \text{ starting on day } d \\ 1 \text{ if } t \text{ is in the first } ebs \text{ time periods of the duties of shift } s \text{ starting on day } d \\ 1 \text{ if } t \text{ is in the last } wpml\text{-1 time periods of the duties of shift } s \text{ starting on day } d \\ 1 - R_s^* \quad \text{otherwise} \end{cases} \quad (71)$$

In Figure 4 we have plotted the number of demand fulfilled by a shift, as well as the corresponding virtual shift, starting at 09:00, ending at 17:00 and having 10 duties. For this example we use that ebs = 30 minutes and $wpml$ = 30 minutes. Furthermore we take the lunch break to be 30 minutes, the minimum length of a break to be 10 minutes, and

Figure 4: Shift and Virtual Shift



the shift is required to have a total break time of 80 minutes. Taking a time period of 5 minutes, we find that $InactivePeriods$ is equal to 14 . Note that the demand fulfilled by the two is equal for the first few and last few time periods of the shift, but the virtual shift has a lower number of active workers in the rest of the periods. This is to account for break time which has to be scheduled.

While finding solutions we noticed that by choosing a lower value for the number of inactive periods we obtained better results. More specifically be using IP^* for the inactive periods instead, where

$$IP^* = \textbf{total\_break\_time} \tag{72}$$

Later we will show results for both definitions of the inactive periods and argue their difference. After we have shown results for shifts and breaks design we will also propose an improvement for the method which is based on the different results obtained by both definitions of inactive periods.

6.2 Algorithm Shifts and Breaks Design

Our solution approach to the shifts and breaks design problem consists of two phases. In the first phase we formulate an ILP model of shift design, as described in section 4.1, using the virtual shifts as described in Section 6.1. From the solution of this resulting ILP a set of shifts and number of duties per shift for each day is constructed. In the second phase of our algorithm we find a break allocation for each duty. Hereto we use Algorithm 3. Pseudocode for the full algorithm to the shits and breaks design problem is given in Algorithm 4.

First Phase
  Construct virtual shift for each possible shift
  Solve shift design for virtual shifts
  Construct initial solution with shifts according to virtual shifts
Second Phase
  Initialise $Active\_Workers$ using E (assuming no breaks)
  $Current\_Solution = Obj(E)$.
  $Previous\_Solution = \infty$;
  $F_{t,e} = 1 \quad \forall t, e$
  while *Current_Solution*<*Previous_Solution* **do**
    for $e \in E$ **do**
      if $\exists \quad e' \neq e$ *s.t.* $F_{t,e'} = 1$ ***and*** $t \in e$ **then**
        if *e has a break allocation* **then**
          remove break allocation from e
          update $Active\_Workers$
        end
        Solve Break Scheduling ILP for duty e using $Active\_Workers$
        Use solution from ILP to allocate breaks to e
        Update $Active\_Workers$
        Update $F_{t,e} \quad \forall t$.
      end
    end
    $Previous\_Solution$=$Current\_Solution$
    $Current\_Solution$=$Obj(E)$.
  end

Algorithm 4: Shifts and Breaks Design Algorithm

6.3 Problem Instances

We have used three sets of instances to test our algorithm. All of the instances are available at [12]. The first two sets contain randomly generated instances. The shift types for these instances is identical to the shift types given for the shift design problem and hence shown in Table 5. The parameters for break scheduling are identical to parameters used for the break scheduling instances are given in Table 8. For instances of the first two sets there is a 'best known' solution which was used to construct the staffing requirements and therefore the best known solution provides exact coverage of the requirements. Furthermore the penalty for the number of shifts is taken to be $W_3 = 60$. According to Di Gaspero et al. [15]: "these weights were selected based on experience with solving problems in real life applications."

The third set of instances are based on a real world example. For these instances a solution in which there is no overstaffing and understaffing is not guaranteed to exist. These instances are also publicly available at [12]. The real life instances are smaller than the randomly generated instances judging by the total staffing requirements. For the real life case the average value for the sum of the staffing requirements over all time periods was equal to 10193, for the randomly generated instances this number was 16535. However the given shift types for the real life instances allowed for more possible shifts. These shift types are given in 10. Other parameters are equal to those used for the randomly generated instances. For these instances the total number of possible shifts is equal to 8645. Also the number of feasible break allocations for breaks can be higher since the shift types allow for shifts longer than shifts in the randomly generated instances. As for the smaller shifts, those lasting 5 hours have $17,084$ feasible break allocations and shifts lasting 6 hours have $114,321$. Shifts lasting 6:05 hours only have 955 different break allocations, this number is much lower since these are the shortest shifts requiring a lunch break and there is not a lot of flexibility possible for the break allocation. We also tried to find the number of feasible break allocations for long shifts. However after 30 minutes Cplex found over $6,000,000$ possible break allocations for shifts lasting 10 hours, which is equal to the lowerbound found for shifts lasting 9 hours.

Due to the higher number of possible shifts we were not able to find satisfactory results when solving an ILP containing all possible shifts in the first phase. In order to solve this issue we only considered shifts at a time granularity of 15 minutes for the real life instances. This brings the number of possible shifts from 8645 to 1075 shifts.

Table 10: Shift Types for Real Life Instances

Abbr.	Name	Earliest Start	Latest Start	Minimum Length	Maximum Length
F	Earlyshift	05:30	08:00	06:00	12:45
T	Dayshift	10:00	12:30	07:00	12:00
S	Lateshift	14:00	17:30	05:00	09:00
N	Nightshift	20:00	23:00	08:00	12:00

6.4 Results

The algorithm for shifts and breaks design was written in C++ using Microsoft Visual Studio 2013 [9]. The integer linear programs were solved using the commercially available optimisation package Cplex [10]. The algorithm was performed on a PC with an Intel i7-4702MQ quad core processor with 2.2Ghz and 8GB RAM memory. Di Gaspero et al. [15] used a time limit of 60 minutes to obtain their results for the shifts and breaks design problem, and for these results we allowed our algorithm a maximum of 30 minutes for the first phase and no time limit for the second phase.

In this section we will show results for the randomly generated instances. Afterwards we will propose an improvement for the first phase of our solution approach and show results obtained for the improved method for the randomly generated instances as well as results for the real life instances. Tables 11 and 12 show results for the first and second set of data. For these results we used two versions of Algorithm 4. Recall that Algorithm 4 uses the virtual shifts in the first phase, and in the second phase the break scheduling ILP for a single duty is used as well as the variables $F_{t,e}$ to skip unnecessary break scheduling ILP's. For the first version of this algorithm the virtual shifts are constructed where the number of breaks is included while calculating the inactive periods. In the tables this method is indicated by the column header 'yes'. For the other algorithm the virtual shifts were constructed by only counting the **total_break_time** as inactive periods. This approach is indicated by 'No'.

For each of our approaches we show the total overstaffing, understaffing and number of shifts used. We also show the time taken to reach our results in minutes. Note that we set a time limit of 30 minutes for the first phase. For each instance we also report the objective value as found by Di Gaspero et al. [15] as indicated by the column 'Hybrid'. Note that the authors did not report results for $1-11$ and $2-11$. The Best Known solutions are the solutions used for constructing the staffing requirements of each instance. Therefore these solutions have no penalty cost on the number of overstaffing and understaffing (since those are 0) and only penalise the number of shifts used. The results show that the 'No' approach outperforms the 'Yes' approach on all instances and both approaches outperform

the method by Di Gaspero et al. [15] on all instances.

The 30 minutes were used in all instances for the first phase. No optimal solution was found (or proven) in this phase. The average optimality gap was 0.34 (SD=0.08) for the 'No' approach and 0.37 (SD=0.08) for the 'yes' approach.

6.5 Improvement

Based on results obtained using Algorithm 4 we propose an improvement to it. We will first describe the reasoning for this improvement and then we will formulate an update to the ILP used in the first phase. We will use this modified ILP to obtain new results to the randomly generated instances as well as the real life instances.

The results showed that the 'Yes' leads to less understaffing. However the overstaffing increases substantially, so much that the objective value for 'Yes' is worse than the 'No' approach. The reason for this is that the virtual shifts used in the shift design ILP have no flexibility. Instead the number of active workers at each time period is fixed based on the selected shifts. Therefore there are time periods in which the ILP chooses to use a higher number of duties than necessary. In the shift design ILP, only the number of duties active according to the virtual shifts can be considered while in the shifts and breaks design problem it is possible to not plan any breaks at a time period and hence have more active workers than calculated by the virtual shifts.

In the first phase we use the virtual shifts to account for the break time which has to be scheduled. However we assume that all break time will be scheduled uniformly across the interval of a duty. In the shifts and breaks design problem it is possible to schedule breaks in a way to handle fluctuations in demand. In order to allow for better solutions we will modify the ILP of Section 4.1 such that the number of active workers at each time period is not completely fixed.

We introduce parameters $A^*_{s,d,t}$. Their value will be equal to the parameters for the actual shifts (instead of virtual shifts). Therefore we can define them in terms of $A_{s,d,t}$ as follows.

$$A^*_{s,d,t} = \begin{cases} 0 & \text{if } A_{s,d,t} = 0 \\ 1 & \text{if } A_{s,d,t} = 1 \\ 1 & \text{otherwise} \end{cases} \tag{73}$$

The constraint specifying the understaffing is rewritten to

$$\sum_{s,d} \left(A^*_{s,d,t} \cdot w_{d,s}\right) + u_t \geq R_t \quad \forall t \tag{74}$$

Table 11: Results Shifts and Breaks Design First Set

Instance	Overstaffing		Understaffing		Shifts		Objective				Time (mins)	
	No	Yes	No	Yes	No	Yes	No	Yes	Hybrid	Best	No	Yes
1-1	1025	1979	86	16	11	20	3570	5318	10540	480	45	40
1-2	1714	2567	141	65	31	47	6698	8604	14904	600	44	40
1-3	1077	2221	115	34	17	36	4324	6942	15330	600	51	43
1-4	1680	3281	203	18	68	81	9470	11602	18652	960	56	49
1-5	1270	2140	100	14	15	28	4440	6100	11656	480	43	39
1-6	934	1564	93	30	17	13	3818	4208	8756	420	37	35
1-7	1117	2008	97	15	23	35	4584	6266	10042	540	43	40
1-8	1457	2592	176	19	39	62	7014	9094	14210	600	49	43
1-9	1092	1983	118	41	27	24	4984	5816	12120	600	43	42
1-10	1583	2511	165	16	55	53	8116	8662	15804	660	54	54
1-11	916	1421	24	4	5	7	2372	3302	n.a.	120	36	35
1-12	919	1573	103	44	11	13	3528	4366	8360	360	47	47
1-13	1355	2242	116	35	16	21	4830	6094	12306	420	51	46
1-14	1337	3394	263	17	72	82	9624	11878	18146	780	84	74
1-15	856	998	7	5	5	5	2082	2346	4774	180	32	33
1-16	1361	2782	134	38	60	61	7662	9604	15820	900	59	63
1-17	1357	3165	167	15	86	78	9544	11160	18402	1080	106	92
1-18	1626	2797	75	8	42	48	6522	8554	16668	720	53	95
1-19	778	2305	174	33	44	39	5936	7280	13582	720	63	97
1-20	1768	2765	112	82	27	29	6276	8090	16794	540	51	103
1-21	1334	1695	79	57	21	23	4718	5340	10188	480	40	73
1-22	770	1314	133	30	5	6	3170	3288	9816	300	45	38
1-23	1594	2916	69	23	48	57	6758	9482	13626	600	48	40
1-24	1134	2117	107	33	30	23	5138	5944	11730	480	42	39
1-25	1110	2918	223	29	63	64	8230	9966	18436	960	58	47
1-26	1336	2593	133	26	47	42	6822	7966	16286	660	48	44
1-27	1457	2903	109	20	15	20	4904	7206	18484	480	61	53
1-28	1587	2065	80	35	22	24	5294	5920	9952	540	39	38
1-29	1260	2521	147	29	52	57	7110	8752	13646	720	57	47
1-30	905	1674	105	13	13	12	3640	4198	8604	300	36	36
Average	1257	2300	122	28	33	37	5706	7112	13367	576	51	52

Table 12: Results Shifts and Breaks Design Second Set

Instance	Overstaffing		Understaffing		Shifts		Objective				Time (mins)	
	No	Yes	No	Yes	No	Yes	No	Yes	Hybrid	Best	No	Yes
2-1	1424	2246	182	56	49	52	7244	8172	14002	720	51	47
2-2	1647	2723	95	12	38	50	6524	8566	12866	720	60	53
2-3	1207	2933	165	95	69	63	8204	10596	13858	720	72	67
2-4	1231	2343	162	35	35	36	6182	7196	12780	720	53	54
2-5	1520	2743	102	15	47	45	6880	8336	12962	720	66	66
2-6	1592	2888	170	50	59	63	8424	10056	16214	720	83	81
2-7	1437	2835	138	62	56	55	7614	9590	17044	720	92	45
2-8	1440	2811	228	37	33	48	7140	8872	13684	720	83	41
2-9	1628	2708	102	101	34	49	6316	9366	14932	720	50	45
2-10	1161	2761	191	15	56	65	7592	9572	17972	720	65	52
2-11	1289	2897	163	40	66	65	8168	10094	n.a.	960	54	44
2-12	1133	2870	159	28	74	69	8296	10160	16028	960	55	46
2-13	1147	2489	257	17	79	92	9604	10668	17446	960	68	47
2-14	1164	3003	298	50	55	74	8608	10946	18636	960	65	48
2-15	1278	2526	253	38	67	76	9106	9992	19032	960	58	53
2-16	1068	2702	257	21	68	74	8786	10054	18950	960	70	58
2-17	1063	3219	308	43	61	35	8866	8968	15754	960	60	55
2-18	1304	3035	221	31	79	84	9558	11420	18616	960	63	56
2-19	1214	2544	220	44	72	82	8948	10448	19456	960	69	61
2-20	1415	3186	148	14	65	78	8210	11192	18688	960	70	59
2-21	1286	2781	228	16	95	106	10552	12082	18890	1200	89	75
2-22	1463	3556	240	24	84	88	10366	12632	19804	1200	84	77
2-23	1304	3126	232	60	93	99	10508	12792	17236	1200	72	71
2-24	1250	3222	162	70	80	77	8920	11764	18178	1200	78	50
2-25	1455	3358	237	48	85	91	10380	12656	19198	1200	100	49
2-26	1346	2756	258	30	74	83	9712	10792	19662	1200	117	53
2-27	1636	3767	235	37	83	99	10602	13844	20200	1200	78	56
2-28	1128	3097	208	37	84	90	9376	11964	16414	1200	63	47
2-29	1400	3038	241	12	77	83	9830	11176	18574	1200	70	55
2-30	1139	3650	457	33	89	110	12188	14230	24462	1200	94	62
Average	1326	2927	211	39	67	73	8757	10607	17294	960	72	56

It follows that we only count understaffing in the case that there are not enough present workers. In such a case understaffing in the shifts and breaks design problem would be inevitable. The overstaffing needs to be changed slightly to avoid the overstaffing becoming negative. Note that we use the parameters corresponding to virtual shifts for the next constraint.

$$o_t \geq \sum_{s,d} \left(A_{s,d,t} \cdot w_{d,s}\right) + u_t - R_t \qquad \forall t \tag{75}$$

$$\tag{76}$$

Therefore we only count overstaffing if there are too many employees using the virtual shifts. This updated ILP considers there to be no overstaffing nor understaffing if the number of present workers is between the number specified by the shifts and the virtual shifts.

A problem in this formulation can be that overall too few present workers are scheduled since the ILP allows for each duty to always be working. In order to solve this issue we enforce that over a certain number of time periods the sum of requirements are for filled using the parameters $A_{s,d,t}$ as specified by the virtual shifts. Hereto we use the parameter C. This will indicate the number of consecutive time periods over which the sum of requirements must be satisfied using the virtual shifts. The following constraint forces this. Note that we abuse notation by allowing the index $t + i$ to be greater than the number of time periods (n). Since the first time period follows the last time period $t + i$ should actually be $(t + i)$ **mod** n

$$\sum_{i=1}^{C} \left(\sum_{s,d} A_{s,d,t+i} \cdot w_{d,s}\right) \geq \sum_{i=1}^{C} R_{t+i} \quad \forall t \tag{77}$$

6.6 Improved Results

In this section we will show results to the shifts and breaks design problem using the techniques discussed in the previous section. After some initial testing we decided to use $C = 25$ as it gave the best results. In Table 14 we show results of this improved algorithm, shown in the columns called '2P-ILP', from two-phase integer linear programming. We compare the results to results obtained without the improvement, indicated by the term 'Prev'. For comparison results of Di Gaspero et al. [15] and the best known solutions are also shown. The running time (in minutes) of both our approaches is also shown. Note that we did not set a time limit for the second phase in the approach 'Prev' but for the '2P-ILP' approach we set a maximum time limit of 30 minutes for the second phase. Table 15 shows results for instances of the second set.

We have also applied our improved algorithm to the real-life instances. As mentioned when describing the problem instances we only considered possible shifts with a time granularity of 15 minutes as opposed to the 5 minutes specified by the instances. Results are shown in Table 13. We compare our results, as indicated by '2P-ILP', to the results of Di Gaspero et al. [15], as indicated by 'Hybrid'. For each of the 5 instances we report the overstaffing, understaffing, number of shifts and the objective value for both approaches. We also show the number of minutes it took for our two-phase approach to reach the solutions. The results of Di Gaspero et al. [15] were reached by allowing a running time of 60 minutes. We allowed our algorithm a maximum time of 30 minutes for each phase.

During the first phase the maximum time of 30 minutes was always reached. The average optimality gap for the randomly generated instances was 0.50 (SD=0.10), for the real life instances this number was 0.86 (SD=0.02). For 20 out of the 60 instances the algorithm of the second phase could not be completed within 30 minutes. The average number of iterations done by the break allocation algorithm was 3.7 (SD=0.85) for the randomly generated instances and 3.6 (SD=1.02) for the real life instances.

We find further improvements for 57 out of the 60 randomly generated instances. On average our obtained solutions are over 7 times as much as the best known solution. The results even show that for all of the randomly generated instances we use more shifts than the best known solution.

Table 13: Results: Shifts and Breaks Design - Real Life

Instance	Overstaffing		Understaffing		Shifts		Objective		Time
	Hybrid	2P-ILP	Hybrid	2P-ILP	Hybrid	2P-ILP	Hybrid	2P-ILP	2P-ILP
2fc04a	2636	1224	173	5	23	30	8382	**4298**	56
3fc04a	2732	1227	130	1	21	36	8024	**4624**	57
4fc04a	2710	1081	94	4	21	32	7620	**4122**	56
50fc04	2636	1130	180	4	29	29	8812	**4040**	**60**
51fc04	2890	1343	209	0	23	32	9250	**4606**	**60**
Average	2720	1201	157	3	23	32	8418	**4338**	58

Table 14: Results Shifts and Breaks Design 2P-ILP Set 1

Instance	Overstaffing		Understaffing		Shifts		Objective				Time (mins)	
	Prev	2P-ILP	Prev	2P-ILP	Prev	2P-ILP	Prev	2P-ILP	Hybrid	Best	Prev	2P-ILP
1-1	1025	1237	86	3	11	13	3570	**3284**	10540	480	45	42
1-2	1714	1740	141	2	31	31	6698	**5360**	14904	600	44	40
1-3	1077	1129	115	7	17	13	4324	**3108**	15330	600	51	48
1-4	1680	2215	203	16	68	47	9470	**7410**	18652	960	56	56
1-5	1270	1537	100	10	15	22	**4440**	4494	11656	480	43	42
1-6	934	1005	93	4	17	13	3818	**2830**	8756	420	37	37
1-7	1117	1716	97	12	23	22	**4584**	4872	10042	540	43	48
1-8	1457	1715	176	4	39	33	7014	**5450**	14210	600	49	53
1-9	1092	1470	118	6	27	24	4984	**4440**	12120	600	43	51
1-10	1583	1895	165	8	55	38	8116	**6150**	15804	660	54	**60**
1-11	916	948	24	0	5	3	2372	**2076**	n.a.	120	36	39
1-12	919	1132	103	8	11	10	3528	**2944**	8360	360	47	51
1-13	1355	1494	116	7	16	14	4830	**3898**	12306	420	51	**60**
1-14	1337	2035	263	5	72	44	9624	**6760**	18146	780	84	**60**
1-15	856	1059	7	1	5	7	**2082**	2548	4774	180	32	34
1-16	1361	2023	134	12	60	39	7662	**6506**	15820	900	59	48
1-17	1357	2079	167	7	86	52	9544	**7348**	18402	1080	106	59
1-18	1626	1824	75	10	42	28	6522	**5428**	16668	720	53	58
1-19	778	1246	174	12	44	36	5936	**4772**	13582	720	63	**60**
1-20	1768	1734	112	11	27	21	6276	**4838**	16794	540	51	**60**
1-21	1334	1523	79	3	21	22	4718	**4396**	10188	480	40	43
1-22	770	782	133	2	5	6	3170	**1944**	9816	300	45	37
1-23	1594	1851	69	14	48	33	6758	**5822**	13626	600	48	45
1-24	1134	1483	107	11	30	23	5138	**4456**	11730	480	42	42
1-25	1110	1757	223	10	63	44	8230	**6254**	18436	960	58	54
1-26	1336	1622	133	8	47	39	6822	**5644**	16286	660	48	48
1-27	1457	1746	109	21	15	11	4904	**4362**	18484	480	61	**60**
1-28	1587	1579	80	4	22	20	5294	**4398**	9952	540	39	40
1-29	1260	1813	147	12	52	36	7110	**5906**	13646	720	57	51
1-30	905	1444	105	0	13	8	3640	**3368**	8604	300	36	39
Average	1257	1561	122	8	33	25	5706	4702	13367	576	51	49

Table 15: Results Shifts and Breaks Design 2P-ILP Set 2

Instance	Overstaffing		Understaffing		Shifts		Objective				Time (mins)	
	Prev	2P-ILP	Prev	2P-ILP	Prev	2P-ILP	Prev	2P-ILP	Hybrid	Best	Prev	2P-ILP
2-1	1424	1485	182	13	49	38	7244	**5380**	14002	720	51	52
2-2	1647	1792	95	3	38	33	6524	**5594**	12866	720	60	**60**
2-3	1207	1982	165	45	69	38	8204	**6694**	13858	720	72	**60**
2-4	1231	1501	162	6	35	37	6182	**5282**	12780	720	53	58
2-5	1520	1610	102	8	47	35	6880	**5400**	12962	720	66	**60**
2-6	1592	1678	170	58	59	46	8424	**6696**	16214	720	83	**60**
2-7	1437	1603	138	19	56	57	7614	**6816**	17044	720	92	49
2-8	1440	1848	228	2	33	34	7140	**5756**	13684	720	83	44
2-9	1628	1720	102	15	34	42	6316	**6110**	14932	720	50	47
2-10	1161	1868	191	7	56	40	7592	**6206**	17972	720	65	58
2-11	1289	1844	163	8	66	48	8168	**6648**	n.a.	960	54	48
2-12	1133	1786	159	3	74	58	8296	**7082**	16028	960	55	52
2-13	1147	1704	257	2	79	58	9604	**6908**	17446	960	68	57
2-14	1164	1976	298	8	55	42	8608	**6552**	18636	960	65	**60**
2-15	1278	2104	253	24	67	47	9106	**7268**	19032	960	58	**60**
2-16	1068	1785	257	20	68	60	8786	**7370**	18950	960	70	**60**
2-17	1063	1787	308	7	61	55	8866	**6944**	15754	960	60	53
2-18	1304	1871	221	16	79	52	9558	**7022**	18616	960	63	**60**
2-19	1214	1935	220	5	72	51	8948	**6980**	19456	960	69	**60**
2-20	1415	1988	148	16	65	52	8210	**7256**	18688	960	70	54
2-21	1286	2147	228	7	95	74	10552	**8804**	18890	1200	89	**60**
2-22	1463	1869	240	5	84	59	10366	**7328**	19804	1200	84	58
2-23	1304	2000	232	25	93	69	10508	**8390**	17236	1200	72	58
2-24	1250	1774	162	8	80	60	8920	**7228**	18178	1200	78	59
2-25	1455	1854	237	12	85	60	10380	**7428**	19198	1200	100	**60**
2-26	1346	2168	258	17	74	59	9712	**8046**	19662	1200	117	**60**
2-27	1636	2324	235	34	83	66	10602	**8948**	20200	1200	78	**60**
2-28	1128	1953	208	1	84	65	9376	**7816**	16414	1200	63	**60**
2-29	1400	2105	241	54	77	63	9830	**8530**	18574	1200	70	**60**
2-30	1139	1996	457	6	89	66	12188	**8012**	24462	1200	94	58
Average	1326	1869	211	15	67	52	8757	**7016**	17294	960	72	57

7 Conclusion

In this thesis we proposed an algorithm for the shifts and breaks design problem. The algorithm improves results found in the literature. Our approach splits the problem up into two different parts. First we find a set of shifts and a number of duties for shifts on each day. Second we allocate breaks to each duty. During the first phase we obtain a solution to an integer linear program which has an average optimality gap of 0.50 for the randomly generated instances and 0.86 for the real life instances. Our algorithm for allocating breaks shows to be effective in allocating breaks to the shifts and duties obtained from this solution since a solution to the shifts and breaks design problem with little understaffing is found.

From the real life examples we have seen that the number of possible shifts to be considered in the first phase grows too large Cplex will not manage to find a (satisfactory) solution. We proposed to solve this issue by only considering all possible shifts using a time granularity of 15 minutes as opposed to 5 minutes. It is possible to apply a similar method to other instances in which the number of possible shifts is too large.

Furthermore we tested the effectiveness of integer linear programming for shift design and for break scheduling. ILP showed to be effective for shift design as we were able to find (and prove) optimal solutions to 54 out of the 60 tested instances. For the other 6 instances we improved the best known solutions as well as proving an upperbound to the problems (using the gap as reported by Cplex). Furthermore we tested the effectiveness of Cplex in obtaining fast solutions and showed that Cplex found better solutions than the GrMCMF+LS method proposed by Di Gaspero et al. [14].

We were not able to find exact solutions to the break scheduling problem using integer linear programming. The reasoning for this is the large number of feasible break allocations allowed per duty. We proposed an algorithm which optimally allocates the breaks for a single duty. This algorithm was outperformed by the memetic algorithm of Widl and Musliu [33]. Since the algorithm was fully completed for some runs of break scheduling we also show that the local minimum reached by our algorithm is unlikely to be a global minimum.

Since we were not able to solve the ILP in the first phase to optimality we are not sure of the performance of our algorithm in case the optimal solution for this ILP would be found. Another issue with our approach is that it can take a long time to obtain a feasible solution. We were able to find a solution to all instances within 60 minutes however this might not be true for larger instances.

For the randomly generated instances, for which a solution with exact coverage is known, our solution costs where still more than 5 times greater than the cost of the solution having exact coverage. In order to further improve results to the shifts and breaks desing problem it might be worthwhile to combine ILP methods with other methods such as local search. As an example the usage of integer linear programming on instances of break scheduling showed to be less effective than the use of memetic algorithms. Furthermore we did not use the break allocation to change the selected shifts. After breaks have been scheduled it could be possible to use this information to add or remove a shift or duty.

8 References

[1] Krzysztof Apt. *Principles of Constraint Programming*. Cambridge University Press, New York, NY, USA, 2003.

[2] Turgut Aykin. Optimal shift scheduling with multiple break windows. *Management Science*, 42(4):591–602, 1996.

[3] Turgut Aykin. A comparative evaluation of modeling approaches to the labor shift scheduling problem. *European Journal of Operational Research*, 125(2):381 – 397, 2000.

[4] John J. Bartholdi, James B. Orlin, and H. Donald Ratliff. Cyclic scheduling via integer programs with circular ones. *Operations Research*, 28(5):1074–1085, 1980.

[5] Stephen E. Bechtold and Larry W. Jacobs. Implicit modeling of flexible break assignments in optimal shift scheduling. *Management Science*, 36(11):1339–1351, 1990.

[6] Andreas Beer, Johannes Gartner, Nysret Musliu, Werner Schafhauser, and Wolfgang Slany. An ai-based break-scheduling system for supervisory personnel. *IEEE Intelligent Systems*, 25 (2):60–73, 2010.

[7] Alex Bonutti, Sara Ceschia, Fabio De Cesco, Nysret Musliu, and Andrea Schaerf. Modeling and solving a real-life multi-skill shift design problem. *Annals of Operations Research*, pages 1–18, 2016.

[8] Gerhard Brewka, Thomas Eiter, and Miroslaw Truszczynski. Answer set programming at a glance. *Commun. ACM*, 54(12):92–103, December 2011.

[9] Microsoft Corperation. Microsoft visual studio community 2013, version 12.031101.00 update 4.

[10] IBM Corporation. Ibm ilog cplex optimization studio 12.7.1.

[11] George B. Dantzig. A comment on Edie's "Traffic Delays at Toll Booths". *Journal of the Operations Research Society of America*, 2(3):339–341, 1954.

[12] Database and Vienna University of Technology Artificial Intelligence Group. Shift design and break scheduling benchmarks. URL `http://www.dbai.tuwien.ac.at/proj/SoftNet/Supervision/Benchmarks/`. Accessed: 04-04-2017.

[13] Rina Dechter, Itay Meiri, and Judea Pearl. Temporal constraint networks. *Artificial Intelligence*, 49(1):61 – 95, 1991. ISSN 0004-3702.

[14] Luca Di Gaspero, Johannes Gärtner, Guy Kortsarz, Nysret Musliu, Andrea Schaerf, and Wolfgang Slany. The minimum shift design problem. *Annals of Operations Research*, 155(1):79–105, 2007.

[15] Luca Di Gaspero, Johannes Gärtner, Nysret Musliu, Andrea Schaerf, Werner Schafhauser, and Wolfgang Slany. A hybrid LS-CP solver for the shifts and breaks design problem. In *International Workshop on Hybrid Metaheuristics*, pages 46–61, Heidelberg, 2010. Springer.

[16] Luca Di Gaspero, Johannes Gärtner, Nysret Musliu, Andrea Schaerf, Werner Schafhauser, and Wolfgang Slany. Automated shift design and break scheduling. In *Automated scheduling and planning*, pages 109–127. Springer, 2013.

[17] Leslie C. Edie. Traffic delays at toll booths. *Journal of the operations research society of America*, 2(2):107–138, 1954.

[18] A.T. 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.

[19] Fred Glover and Manuel Laguna. *Tabu Search*, pages 2093–2229. Springer US, Boston, MA, 1999.

[20] Georg Gotlob, Slany Wolfgang, and Musliu Nysret. Rota: A research project on algorithms for workforce scheduling and shift design optimiation. http://www.dbai.tuwien.ac.at/proj/Rota/benchmarks.html. Accessed: 04-04-2017.

[21] Dorit S. Hochbaum and Asaf Levin. Cyclical scheduling and multi-shift scheduling: Complexity and approximation algorithms. *Discrete Optimization*, 3(4):327 – 340, 2006.

[22] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi. Optimization by simulated annealing. *Science*, 220(4598):671–680, 1983.

[23] Guy Kortsarz and Wolfgang Slany. The minimum shift design problem and its relation to the minimum edge-cost flow problem. Technical Report DBAI-TR-2001-46, Technische Universität Wien, June 2001. URL `http://www.dbai.tuwien.ac.at/staff/slany/pubs/dbai-tr-2001-46.pdf`.

[24] Steven Minton, Mark D. Johnston, Andrew B. Philips, and Philip Laird. Minimizing conflicts: a heuristic repair method for constraint satisfaction and scheduling problems. *Artificial Intelligence*, 58(1):161 – 205, 1992.

[25] Pablo Moscato. On evolution, search, optimization, genetic algorithms and martial arts - towards memetic algorithms, 1989.

[26] K. Mulmuley, U.V. Vazirani, and V.V. Vazirani. Matching is as easy as matrix inversion. *Combinatorica*, 7(1):105–113, 1987.

[27] N. Musliu, J. Grtner, and W. Slany. Efficient generation of rotating workforce schedules. *Discrete Applied Mathematics*, 118(1-2):85–98, 2002.

[28] Nysret Musliu, Andrea Schaerf, and Wolfgang Slany. Local search for shift design. *European Journal of Operational Research*, 153(1):51 – 64, 2004.

[29] Nysret Musliu, Werner Schafhauser, and Magdalena Widl. A memetic algorithm for a break scheduling problem. In *8th Metaheuristic International Conference, Hamburg, Germany*, 2009.

[30] Monia Rekik, Jean-Franois Cordeau, and Franois Soumis. Implicit shift scheduling with multiple breaks and work stretch duration restrictions. *J. Scheduling*, 13:49–75, 2010.

[31] Jorne Van den Bergh, Jeroen Beliën, Philippe De Bruecker, Erik Demeulemeester, and Liesje De Boeck. Personnel scheduling: A literature review. *European Journal of Operational Research*, 226(3):367 – 385, 2013.

[32] Arthur F. Veinott and Harvey M. Wagner. Optimal capacity scheduling-i. *Operations Research*, 10(4):518–532, 1962.

[33] Magdalena Widl and Nysret Musliu. *An Improved Memetic Algorithm for Break Scheduling*, pages 133–147. Springer Berlin Heidelberg, Berlin, Heidelberg, 2010.

[34] Magdalena Widl and Nysret Musliu. The break scheduling problem: complexity results and practical algorithms. *Memetic Computing*, 6(2):97–112, 2014.

[35] Ximes. Operating hours assistant 4.1. URL `http://www.ximes.com/en/products/software/operating-hours-assistant-opa/`. Accessed: 30-10-2017.

9 Appendix

9.1 ILP Shift Design

Sets

Time Periods	t
days	d
shifts	s

Variables

a_s	Binary variable indicating whether shift s is active
$w_{d,s}$	Integer variable indicating how many workers will be working on shift s on day d
u_t	Will denote the understaffing at time period t
o_t	Will denote the overstaffing at time period t

Parameters

R_t	The requirement at time period t
$A_{s,d,t}$	Binary, indicating if the duty starting on day d of shift s is active on time period t
W_1	Penalty cost for overstaffing
W_2	Penalty cost for understaffing
W_3	Penalty cost for the number of shifts
M	Is used as a large constant, here it is the maximum demand at any time period

Constraints

$$w_{d,s} \leq M \cdot a_s \qquad \forall d, s \tag{78}$$

$$\sum_{s,d} \left(A_{s,d,t} \cdot w_{d,s}\right) + u_t \geq R_t \qquad \forall t \tag{79}$$

$$o_t = \sum_{s,d} \left(A_{s,d,t} \cdot w_{d,s}\right) + u_t - R_t \qquad \forall t \tag{80}$$

$$a_s \in \{0, 1\} \qquad \forall s \tag{81}$$

$$w_{d,s} \in \mathbb{N}_0 \qquad \forall d, s \tag{82}$$

$$o_t, u_t \geq 0 \qquad \forall t \tag{83}$$

$$\tag{84}$$

Objective

$$\text{minimize } W_1 \sum_t o_t + W_2 \sum_t u_t + W_3 \sum_s a_s \tag{85}$$

9.2 ILP Break Scheduling Single Duty

Sets

Time Periods	$t = \{1, ..., Length\}$
Time Periods Extra	$t^* = \{1, ..., Length + 1\}$
Breaks	$b = \{1, ..., M_{breaks}\}$

Parameters

D_t	The demand for staffing at time period t
M_{breaks}	The maximum number of different breaks+1
Tbt	The total amount of break time required
$Length$	Length of the duty s (number of time periods)
$Bminl$	Minimum length of a break
$Bmaxl$	Maximum length of a break
$Mlwp$	Minimum long working period
$Lbml$	Long break minimum length
$Wpminl$	Working period minimum length
$Wpmaxl$	Working period maximum length
Ebs	Earliest break start (breaks can start this many time periods after shift start)
Lbs	Latest break start (breaks can start this many time periods from shift end)
L	Binary, indicating whether a lunch break is required
$Elbs$	Earliest lunch break start
$Llbs$	Latest lunch break start
Lbl	Lunch break length
W_1	Penalty cost for overstaffing
W_2	Penalty cost for understaffing
M	Used as a sufficiently large constant. Equal to the number of time periods in the duty

Variables

a_b	Binary variable indicating whether the bth break is active
a_b^l	Binary variable indicating whether break b needs to be long
l_b	Binary variable indicating whether break b is the lunch break
b_b^l	The length (in time slots) that the bth break takes
b_b^s	First time slot on which break b is active
wp_b	Indicates the length of the bth working period
$z^s_{t^*,b}$	Binary indicating if the first time period of the bth break is the t^*th time period
$z^e_{t^*,b}$	Binary indicating if the last time period of the bth break is the t^*th time period
$z_{t,b}$	Binary[1] indicating if the duty is on its bth break during time period t
$z^b_{t,b}$	Binary[1] indicating if time period t is before the start of the bth break
$z^a_{t,b}$	Binary[1] indicating if time period t is after the end of the bth break
x_t	Binary[1] indicating if the duty is working during a time period t
u_t	Understaffing in time period t
o_t	Overstaffing in time period t

[1]These variables can be relaxed to continuous variables on [0,1]. By the binary restriction on $z^s_{t,b}$ and $z^e_{t,b}$ the variables are forced to be either 0 or 1.

Constraints

$$a_b \leq a_{b-1} \qquad \forall b \in B \setminus \{1\} \qquad (86)$$

$$a_b^l + l_b \leq a_b \qquad \forall b \qquad (87)$$

$$\sum_b b_b^l = Tbt \qquad (88)$$

$$M \cdot (1 - a_b) + b_b^l \geq Bminl \qquad \forall b \qquad (89)$$

$$b_b^l \leq Bmaxl \cdot (a_b) + M \cdot (l_b) \qquad \forall b \qquad (90)$$

$$a_b^l \cdot Lbml \leq b_b^l \qquad \forall b \qquad (91)$$

$$b_b^s \geq Ebs \qquad \forall b \qquad (92)$$

$$b_b^s \leq Length - Lbs + (1 - a_b) \cdot (1 + Lbs) \qquad \forall b \qquad (93)$$

$$b_b^s \geq (Length + 1) \cdot (1 - a_b) \qquad \forall b \qquad (94)$$

$$wp_1 = b_1^s - 1 \qquad (95)$$

$$wp_b = b_b^s - (b_{b-1}^s + b_{b-1}^l) \qquad \forall b \in B \setminus \{1\} \qquad (96)$$

$$wp_b \leq Wpmaxl \qquad \forall b \qquad (97)$$

$$wp_1 \geq Wpminl \qquad (98)$$

$$M \cdot (1 - a_{b-1}) + wp_b \geq Wpminl \qquad \forall b \in B \setminus \{1\} \qquad (99)$$

$$wp_b - Mlwp + 1 \leq a_b^l \cdot M + l_b \cdot M + (1 - a_b) \cdot M \qquad \forall b \qquad (100)$$

$$Mlwp - wp_b \leq (1 - a_b^l) \cdot M \qquad \forall b \qquad (101)$$

$$\sum_b l_b = L \qquad (102)$$

$$(1 - l_b) \cdot M \geq Lbl - b_b^l \qquad \forall b \qquad (103)$$

$$(1 - l_b) \cdot M \geq b_b^l - Lbl \qquad \forall b \qquad (104)$$

$$(1 - l_b) \cdot M \geq Elbs - b_b^s \qquad \forall b \qquad (105)$$

$$(1 - l_b) \cdot M \geq b_b^s - Llbs \qquad \forall b \qquad (106)$$

$$\sum_{t^*} z_{t^*,b}^s = 1 \qquad \forall b \qquad (107)$$

$$\sum_{t^*} t^* \cdot z_{t,b}^s = b_b^s \qquad \forall b \qquad (108)$$

$$\sum_{i=1>Length+1} z_{i,b}^s = z_{t,b}^b \qquad \forall t, b \qquad (109)$$

$$\sum_{t^*} z_{t^*,b}^e = 1 \qquad \forall b \qquad (110)$$

$$\sum_{t^*} t \cdot z_{t^*,b}^e = b_b^s + b_b^l - 1 + (1 - a_b) \qquad \forall b \qquad (111)$$

$$\sum_{i=1<t-1} z_{i,b}^e = z_{t,b}^a \qquad \forall t, b \qquad (112)$$

$$z_{t,b} + z_{t,b}^a + z_{t,b}^b = 1 \qquad \forall t, b \qquad (113)$$

$$1 - x_t \geq \sum_b z_{t,b} \qquad \forall t \in T \qquad (114)$$

$$1 - x_t \geq \sum_b z^e_{t-1,b} \qquad \forall t \in T \setminus \{1\} \qquad (115)$$

$$1 - x_1 \leq \sum_b z_{1,b} \qquad (116)$$

$$1 - x_t \leq \sum_b z_{t,b} + \sum_b z^e_{t-1,b} \qquad \forall t \in T \qquad (117)$$

$$x_t - 1 + u_t \geq D_t \qquad \forall t \qquad (118)$$

$$o_t - u_t - x_t + 1 = -D_t \qquad \forall t \qquad (119)$$

$$a_b, a^l_b, l_b \in \{0, 1\} \qquad \forall b \qquad (120)$$

$$z^s_{t,b}, z^e_{t,b} \in \{0, 1\} \qquad \forall t, b \qquad (121)$$

$$o_t, u_t \geq \forall \qquad t \qquad (122)$$

Objective

$$\text{minimize } W_1 \sum_t o_t + W_2 \sum_t u_t \qquad (123)$$