# Exact Methods for Shift Design and Break Scheduling

## DIPLOMARBEIT

zur Erlangung des akademischen Grades

## Diplom-Ingenieur

im Rahmen des Studiums

## Software Engineering & Internet Computing

eingereicht von

### Deniz Kocabas

Matrikelnummer 1127055

an der Fakultät für Informatik
der Technischen Universität Wien

Betreuung: Priv.-Doz. Dr. Nysret MUSLIU

Wien, 23. April 2015

_____          _____
Deniz Kocabas                              Nysret MUSLIU

# Exact Methods for Shift Design and Break Scheduling

## DIPLOMA THESIS

submitted in partial fulfillment of the requirements for the degree of

## Diplom-Ingenieur

in

## Software Engineering & Internet Computing

by

### Deniz Kocabas
Registration Number 1127055

to the Faculty of Informatics
at the Vienna University of Technology

Advisor: Priv.-Doz. Dr.  Nysret MUSLIU

Vienna, 23rd April, 2015

Deniz Kocabas      Nysret MUSLIU

# Erklärung zur Verfassung der Arbeit

Deniz Kocabas
Embelgasse 5-7/6, 1050 Wien

Hiermit erkläre ich, dass ich diese Arbeit selbständig verfasst habe, dass ich die verwendeten Quellen und Hilfsmittel vollständig angegeben habe und dass ich die Stellen der Arbeit – einschließlich Tabellen, Karten und Abbildungen –, die anderen Werken oder dem Internet im Wortlaut oder dem Sinn nach entnommen sind, auf jeden Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht habe.

Wien, 23. April 2015

_____
Deniz Kocabas

# Abstract

Shift design and break scheduling problems are important employee scheduling problems that have been investigated recently in the literature. The shift design problem arises in a variety of large organizations. It involves efficient usage of personnel resources to reduce costs as much as possible, while satisfying several constraints. Break scheduling problem is an important phase in the general employee scheduling in several organizations that needs a high level of concentration. The loss of concentration can end up with a dangerous consequences. Therefore, it is important that the workers have from time to time breaks to keep the concentration level high.

The purpose of the shift design problem is to find a minimum number of legal shifts, that reduce the shortages and excesses of workers in every time slot during the planning period. In the break scheduling, the breaks are assigned within their shifts conveniently with respect to several constraints and also trying to keep the deviation of workforce for the time slots as minimal as possible.

In this thesis, we introduced integer linear programming formulation explicitly for solving shift design and break scheduling problems. The explicit model is investigated based on enumeration of each shift or break from the possible shift or breaks starts and lengths. To solve the integer programming model for the shift design problems from the literature, we performed experiments with state-of-the-art solvers Cplex and Gurobi. The Cplex Solver shows better time performance compared to Gurobi Solver. Our results show that these solvers can be used successfully for the shift design problem. Indeed, new optimal solutions are obtained for several benchmark examples.

Exact method shows also superior performance for break scheduling problem with using Cplex Solver. However, our formulation fails to run in the real life instances of break scheduling problem, due to introduced restrictions in our problem formulation. We improved the previous solutions for several randomly generated examples in break scheduling problem.

# Acknowledgements

# Contents

# List of Figures

# List of Tables

# List of Algorithms

# Introduction

In this thesis, we investigate an integer programming approach to solve two hard problems, shift design and break scheduling. These two problems are important employee scheduling problems and are introduced recently in the literature. The shift scheduling problem has been first introduced by Edie (1954 )[Edi54] in the context of toll booth operators scheduling. The shift scheduling problem today is very different from the one introduced by Dantzig [Dan54] and Edie [Edi54]. The relative importance of needed employees in scheduling decision has grown due to the economic considerations. Part time jobs, flexible work hours, lunch breaks and monitor breaks are the some of reasons to increase research attention. Although several approaches have been used successfully to solve real life problems, the optimal solution for several problems are still not known. Therefore, these problems are still challenging problems.

The shift design problem arises in a variety of large organizations such as airlines, hospitals, telephone companies, police departments, etc. It involves efficient usage of personnel resources to reduce costs as much as possible, while meeting several constraints. The professional planners can construct solution for small practical problems by hand, but for the large number of different demands and solutions, the solution space is too large for an efficient manual approach. Even though finding a solution manually is possible, it is unlikely, that the optimum solution will be found. Furthermore, finding a solution manually usually takes very long time [DGGM+13]. Therefore, different approaches in the literature [MSS04] [DGGK+07] [DGGM+13] [Abs13] have been proposed to solve this problem.

The break scheduling problem is an important phase in the general employee scheduling in several organizations that needs a high level of concentration, such as air traffic control, security checking, supervision, assembly line workers, etc. The loss of concentration in such organization can end up with a dangerous consequences. It is necessary that the workers have from time to time breaks to keep the concentration level high.

In the break scheduling problem, breaks need to be assigned to shifts over one week. The slot length is usually 5 minutes, therefore, huge number of possible assignments of

breaks exist. Due to the problem's size and complexity, to calculate optimum breaks for large number of shifts is impossible. Automatic or computer aided break scheduling is usually the only way to reach high quality shift plans. Therefore, this problem has been considered by researchers in the literature [BGM$^+$08] [BGMS10] [WM14].

The aim of this master's thesis is to investigate new solution techniques for shift design and break scheduling problems. We will introduce an integer programming approach for solving shift design and break scheduling. The integer programming formulation for shift design and break scheduling problem is expected to find optimal solutions for many instances or at least to improve the existing results.

## 1.1 Aim of The Work

The aim of our thesis are:

- Investigate integer linear programming formulation for shift design and break scheduling problems.

- Apply state-of-the-art solvers, Cplex Solver and Gurobi Solver to solve benchmark problems from the literature.

- Compare solutions of our approaches to results of existing state-of-the-art in the literature on benchmark examples.

## 1.2 Results of the Master's Thesis

The main results of this master's thesis are given as follows,

- We propose an integer linear formulation for shift design and break scheduling problems. For break scheduling, due to the large number of constraints and variables, we restricted the problem definition.

- We compared state-of-the-art solvers for shift design problem. In a few instances of shift design problem, we could not obtain results in 2 hours time limit and for these instances we experimented with three different parameters of Cplex Solver. By using these parameters, we can not guarantee that the optimal solution is found. However, we achieved better solutions for these instances by using these parameters. For the break scheduling problem, only the Cplex Solver was applied.

- We compare our algorithms with the best existing results for the both problems in literature. We obtained the best existing result in each instance (except one). However, our formulation fails to run in the real life instances of break scheduling problem, due to introduced restrictions in this problem.

2

## 1.3 Structure of the Master's Thesis

The remaining parts of this thesis are organized into the following chapters:

- In **Chapter 2**, we introduce the shift design and the break scheduling problem. The formal definition of shift design and break scheduling problems, that we have solved in this thesis, are given.

- In **Chapter 3**, we give an overview of state-of-the-art. We discuss the proposed integer linear programming formulations for a similar problem called shift scheduling problem and then we give the related work for shift design and break scheduling problems.

- In **Chapter 4**, we present our integer linear programming formulations for both problems in details. The variables, constraints and objective functions be described.

- In **Chapter 5**, we present the computational results obtained by our integer linear programming formulations for shift design and break scheduling problems. We compared state-of-the-art solvers for shift design problem and experiment different parameters in Cplex Solver for long lasting instances of shift design problem. At last, we compare our results with the best known result in the state-of-the-art for both problems.

- In **Chapter 6**, we conclude this thesis by summarizing the work presented and give ideas for potential future work.

# Problem Statements

In this chapter, the formal definition of shift design and break scheduling problems, that we have solved in this thesis, are given.

## 2.1   Shift Design Problem

Below we give the definition of the shift design problem. Our description is based on the problem definition from [GMS01] [MSS04]:

- Planning period consists of $n$ consecutive time slot $[a_1, a_2), [a_2, a_3), ..., [a_n, a_{n+1})$, all having the same length *slotlength* in minutes. The needed number of workers $w_i$ for each interval $[a_i, a_{i+1})$. The shift design problem has a cyclic structure, therefore the end of the planning period $a_{n+1}$ is equal to the first time point $a_1$. The format of time points is: $day : hour : minute$

- The shifts can be generated depending on $y$ shift types $v_1, ..., v_y$. Each shift type has a minimum/maximum start and minimum/maximum length.

    $v_j.minStart$ : Earliest start of the shift types $j$.

    $v_j.maxStart$ : Latest start of the shift types $j$.

    $v_j.minLength$ : Shortest duration of the shift types $j$.

    $v_j.maxLength$ : Longest duration of the shift types $j$.

    In Table 2.1 an example of four shift types is given.

- In original definition of shift design problem, there are also two scalar real-valued quantities, used to define the distance from the average number of duties.

    AS : The upper limit for the average number of working shifts per week per employee.

| Shift type | MinStart | MaxStart | MinLength | MaxLength |
|:---:|:---:|:---:|:---:|:---:|
| M | 05:00 | 08:00 | 07:00 | 09:00 |
| D | 09:00 | 11:00 | 07:00 | 09:00 |
| A | 14:00 | 16:00 | 07:00 | 09:00 |
| N | 21:00 | 23:00 | 07:00 | 09:00 |

Table 2.1: An example of shift types

AH : Average number of working hours per week per employee.

These two parameters are not used in our integer programming formulation.

The purpose of the shift design problem is to generate $k$ shifts $s_1, s_2, ... s_k$, which belongs to one of the shift types. Every shift has a start point $v_j.start$ and length $v_j.length$ parameters. Additionally, each real shift $s_p$ has adjoined parameters $s_p.w_i, \forall i \in \{1, ..., C\}$ (C represents the number of days in the planning period and usually considered a week) representing the number of workers in shift $s_p$ during the day $i$. The aim is to minimize the four components given below:

$F_1$ : Sum of the excesses of workers in each time slot during the planning period

$F_2$ : Sum of the shortages of workers in each time slot during the planning period

$F_3$ : Number of shifts

$F_4$ : Distance of the average number of duties per week in case it is above a certain threshold. This component is meant to avoid an excessive fragmentation of workers load in too many short shifts.

This is a multi criteria optimization problem. Unlike the four weighted components in the original definition, we have used the first three like in the article [DGGK+07]. These remaining three components have different importance depending on the situation.

The formal representation of the shift design problem is given in detail in the article [GMS01] [MSS04].

## 2.2 Break Scheduling Problem

Below we give the definition of break scheduling problem. Our description is based on the problem definition from [BGMS10]:

- Planning period divided into $n$ consecutive time slot $[a_1, a_2), [a_2, a_3), ..., [a_n, a_{n+1})$, all with the same length *slotlength* usually 5 minutes. The break scheduling problem has also cyclic structure, therefore the last time slot $t_{n+1}$ is equal to the first time slot $t_1$.

- There are $k$ shifts $(s_1, s_2, ..., s_k)$, indicating the work schedule of employees. The break time per shift $s_i$ is calculated based on the shift length. If shift length is less or equal to 10 hours, break time is,

$$s_i.breakTime = floor((minutes(ShiftLength) - 20)/50) * 10 \qquad (2.1)$$

otherwise,

$$s_i.breakTime = ceil(minutes(ShiftLength)/4) \qquad (2.2)$$

- The employee requirements for each time slot, $[a_1, a_2), [a_2, a_3), ..., [a_n, a_{n+1})$ in the planning period are defined as follows,

    - $w_t$ is the needed number of workers for the time slot $t$.
    - A staff is considered to be working for the time slot $t$, if the time slot $t$ is in the employees working schedule and the employees are not in the break period in time slot $t$.
    - An employee needs a full time slot (typically 5 minutes) to return back to work after a break. Thus, the first time slot after the break, the staff is not considered to be working.

- Shifts and breaks have two parameters, that are the beginning or ending time points of shifts or breaks. The duration value can be calculated by substracting the time slots of the *start* and *end* of shifts or breaks. Moreover, each break is associated with a certain shift in which it is scheduled. We distinguish between two different types of breaks, that are monitor or lunch breaks.

Given a planning period, a set of shifts, the associated total break times, and the staffing requirements, there are several hard and soft constraints, that need to be considered for the break scheduling problem. The hard constraints are:

- Each break lies entirely within its associated shift.

$$s_i.start \le b_j.start \le b_j.end \le s_i.end \qquad (2.3)$$

- Two distinct breaks associated with the same shift, do not overlap in time.

$$b_j.start \le b_j.end \le b_k.start \le b_k.end \qquad (2.4)$$

or

$$b_k.start \le b_k.end \le b_j.start \le b_j.end \qquad (2.5)$$

- Sum of break durations needs to be equal to shift's break time.

$$\sum_{b_j \in s_i} b_j.duration = s_i.breakTime \qquad (2.6)$$

7

Among, all feasible solutions for the break scheduling problem, there are seven soft constraints. These constraints are useful to assign the breaks within their shifts conveniently and to reduce the excesses and shortages of workforce for the time slots. These seven criteria are explained below,

$C_1$: Break Positions. Each break, $b_j$, may start, at the earliest, a certain number of time slots after the beginning of its associated shift $s_i$, and may end, at the latest, a given number of time slots before the end of its shift:

$$b_j.start \geq s_i.start + distanceToShiftStart \qquad (2.7)$$

$$b_j.end \leq s_i.end + distanceToShiftEnd \qquad (2.8)$$

$C_2$: Lunch Breaks. A shift $s_i$ can have several lunch breaks, each required to last a specified number of time slots (min lunch break duration), and should be located within a certain time window after the shift start. Let $b_{lb}$ be a lunch break. Then,

$$b_{lb}.start \geq s_i.start + distanceToShiftStartLB \qquad (2.9)$$

$$b_{lb}.end \leq s_i.end + distanceToShiftEndLB \qquad (2.10)$$

$C_3$: Duration of Work Periods. Breaks divide a shift into several work and rest periods. The duration of work periods within a shift must range between a required minimum and maximum duration:

$$minWorkDuration \leq b_1.start - s_i.start \leq maxWorkDuration \qquad (2.11)$$

$$minWorkDuration \leq b_{j+1}.start - b_j.end \leq maxWorkDuration \qquad (2.12)$$

$$minWorkDuration \leq s_i.end - b_m.end \leq maxWorkDuration \qquad (2.13)$$

where $(b_1, ..., b_j, b_{j+1}, ..., b_m)$ are the breaks of $s_i$ in temporal order.

$C_4$: Minimum Break Times after Work Periods. If the duration of a work period exceeds a certain limit, the break following that period must last a given minimum number of time slots ($minTsCount$):

$$b_1.start - s_i.start \geq workLimit \Rightarrow b_1.duration \geq minTsCount \qquad (2.14)$$

$$b_{j+1}.start - b_j.end \geq workLimit \Rightarrow b_{j+1}.duration \geq minTsCount \qquad (2.15)$$

$C_5$: Break Durations. The duration of each break, $b_j$, must lie within a specified minimum and maximum value:

$$minDuration \leq b_j.duration \leq maxDuration \tag{2.16}$$

$C_6$: Shortage of Employees. At least $w_t$ employees should be working in each time interval, $[a_t, a_{t+1})$. Sum of the shortages of workers in each time slot during the planning period indicating the $C_6$.

$C_7$: Excess of Employees. At most $w_t$ employees should be working in each time interval, $[a_t, a_{t+1})$. Sum of the excesses of workers in each time slot during the planning period indicating the $C_7$.

This is also a multi criteria optimization problem. For each soft constraint, we have weight value. These weights can be different depending on the situations. Given an instance of the break scheduling, our goal is to find a feasible solution, that minimizes the violation of soft constraints.

# Related Work

In this section, we will give an overview of state of the art in the area of shift design and break scheduling. The variants of problems, we solved in this thesis are introduced recently in the literature. However a similar problem called shift scheduling problem usually with a break or multiple breaks has been extensively investigated in the literature. We will first give the related work for the shift scheduling problem and then we will give the state of the art for shift design and break scheduling problems.

## 3.1  Shift Scheduling with Breaks

Different approaches have been proposed for the shift scheduling problem, especially based on integer programming formulation. The problem has been first introduced by Edie (1954) [Edi54] in the context of toll booth operator scheduling. Solving this shift scheduling problem was originally proposed by Dantzig [Dan54] by the set covering formulation.

The integer programming model of Dantzig is given below:

$$\min \sum_{j=1}^{n} c_j * x_j \tag{3.1}$$

subject to:

$$\sum_{j=1}^{n} a_{tj} x_j \geq b_t \quad \forall t = 1, 2, ...m \tag{3.2}$$

$$x_j \geq 0, \quad x \in \mathbb{Z} \quad \forall j = 1, 2, ...n \tag{3.3}$$

where,

$n$ : number of possible shifts.

$m$ : number of time slot in the planning period.
$c_j$ : cost of assigning an employee to shift $j$.
$x_j$ : number of workers assigned to shift $j$.
$a_{tj}$ : is 1, if time slot $t$ is a work period for shift $j$, 0 otherwise.
$b_t$ : needed number of employees in time slot $t$.

The solution of this formulation can be found by enumerating the feasible shifts based on possible shift starts, shift durations, breaks, and time windows for breaks. However, involving a high flexibility with including different shift start times, lengths, multiple breaks, multiple break types cause increasing the number of enumerated shifts. For this reason, to solve with the explicit set covering formulation can be very difficult.

Researchers have proposed different formulations to overcome this difficulty. Moondra [Moo76] proposed an approach of implicitly representing shifts. The considered model has two types of shifts; full-time and part-time:

- Full-time shifts: Fixed length and a lunch break window allowing two break starting times. To assign the break placements, half of the employees take their lunch in the first period and the remaining half of workers in the second break period.

- Part-time shifts: With a length variable (4-8 hours) and no lunch break. The length of the part-time shift was represented implicitly.

Bechtold and Jacobs [BJ90] introduced a new integer formulation, that break assignments were modeled implicitly rather than explicitly. The formulation considers a single break with the same duration in each shift. Although the extended formulation was shown to be superior compared to the set covering model, approach is restricted to the less than 24 hours planning period. Thompson [Tho95] combined these two formulation works of Moondra [Moo76] and Bechtold and Jacobs [BJ90] to achieve a fully implicit formulation of the shift scheduling problem. This formulation reduces the size and improve the scheduling flexibility.

Aykin [Ayk96] proposed an implicit integer programming model with LINDO for the shift scheduling problem with break placement. The problem is extended that employees have multiple rest breaks and a lunch break. The length of a lunch break is usually 30 to 60 minutes and a rest break is 15 to 30 minutes. The proposed formulation has also time window for lunch and rest breaks. For instance, Aykin assumed that time window of a break starts half an hour before the ideal break location and take 90 minutes. Therefore, 6 possible different time slots to assign 15 minute break and 5 different ways for a 30 minute break. It reduces the number of variables needed. The approach is not only feasible for less than 24 hours planning period, but also suitable for 24 hours continuous (cyclical problem) planning period.

Aykin [Ayk00] extended Bechtold and Jacobs's formulation [BJ90] a generalized version by relaxing the assumptions of it and compared with the model that he presented in [Ayk96]. Although generalized version of Bechtold and Jacobs's approach has fewer

variables, it has more constaints and more non- zero in A-matrix. Its end up with a worse performance than the model of Aykin.

To overcome the difficulties by explicit set covering formulation, implicit models have been used to solve shift scheduling problems efficiently. However, implicit models use more complex formulations to assign feasible shifts. These approaches have some problems in solving large size problems. This difficulties lead researchers to propose approaches either using branch and price [MMT00] or using branch and cut approach [Ayk98]. Aykin [Ayk98] proposed branch and cut algorithm based on an implicit formulation for the shift scheduling problem. Rounding heuristic is used to add cuts and updated iteratively. Mehrotra et al. [MMT00] developed a branch and price approach. Their formulations obtained good results for the large shift scheduling problems, optimal solutions or best non-optimal solutions were found.

Rekik et al. [RCS10] proposed an implicit formulation of the shift scheduling with multiple breaks. To increase flexibility, fractionable breaks and work stretch duration restriction are used. Each break is appropriately assigned by minimum and maximum pre- and post-break work stretch duration constraints. Breaks can be divided into fractional breaks and this sub breaks can have different lengths, depends on minimum, maximum sub break lengths and the sum of the lengths of sub break is equal to the total break duration of the shift.

## 3.2 Shift Design

The concept of shift design has been first introduced in [GMS01] [MSS04] [Mus01]. There are several differences, that characterize the shift design problem. The shifts are generated over multiple days, usually a week, rather than a day. The shift design problem has a cyclic structure, therefore the last time slot of the week is connected to the first time slot of a week. In order to minimize the number of shifts in shift design problem, we need to consider reusing shifts on all days of the week. Furthermore, the objective function of shift scheduling is to minimize the number of workers, without any shortages of employees.

In the publications above, a local search approach was proposed with a set of move types to explore the neighbourhood. In order to avoid cycles in the move selection process, tabu search mechanism is used. To make the search more effective, the neighbourhood exploration mechanism is based on analyzing the distance of the current solution to the optimal solution with respect to the shortages or excesses of workers with the longest duration and the used shift types. The initial solution of the algorithm is based on every change of the requirements of employees in each time slot. These differences of consecutive time slot can be beginning (increase of requirement) or ending (decrease of requirement) point of a shift.

Di Gaspero et al. [DGGK+07] improved this method and composed a greedy construction heuristic with the local search algorithm. The initial solution is constructed using new greedy heuristic based on min-cost max-flow. Shifts are edges and workforces are the edge flows. In the second stage, the local search paradigm is used to explore the neighbourhood. The hybrid solver outperforms the previous approach. Abseher

[Abs13] proposed different modelling approaches using answer set programming, but the performance could not be improved obtained by solving the shift design problem using the heuristic-based approaches in [DGGK$^+$07].

## 3.3 Break Scheduling

Beer et al. [BGM$^+$08] introduced the break scheduling problem for call centers and developed local search techniques based on min-conflicts algorithm. Although min-conflict search tries to improve the solution incrementally by concentrating on violating constraints, is not being able to escape from local optima. Random walk strategy is adapted to explore further regions with a probability p and the remaining 1-p probability is used for min-conflict search. In [BGMS10] a break scheduling problem for supervisory personnel was investigated, that presented in the previous chapter. Authors proposed tabu search mechanism and simulated annealing algorithm [BGMS10] and the performances compared with min-conflicts-random-walk search. The min-conflicts-random-walk approach outperformed the other two heuristic techniques.

Widl [MSW09] introduced a memetic algorithm for break scheduling problem introduced in [BGMS10]. Initial solutions were constructed randomly or by fast heuristic. Break patterns created and remained unviolated the constraints $C_1$ -$C_5$ during each iteration. Every individual tries to find the best solutions with using a local search algorithm. This approach improved the previous results based on min-conflicts-random-walk for the break scheduling problem.

In [WM14] [Wid10] [WM10] 2 new memetic algorithms were proposed. These approaches are used a new memetic representation based on time periods instead of on each shift, therefore these algorithms require different genetic operators. In the first algorithm, tabu list is used to prevent re-visiting old computed solutions in local search. To improve the first algorithm, a new crossover operator is developed in the second algorithm, that every offspring can have more than one parent and each individual can be parent also. With each iteration, the first offspring is created by joining the best memes of the current memepool and the remaining are created by applying a k-tournament. Bad individuals are less likely to be discarded. To calculate the best memes and to discard some individuals, penalty system was developed. As a result, applying local search for some instances instead of all of them provided a better result. The approach obtain the best results for the break scheduling problem.

## 3.4 Shift Design and Break Scheduling

Generally in the literature, the break scheduling has been addressed mainly as part of the shift scheduling problem. Such an example, which have multiple rest breaks and a lunch break, we presented before, is proposed by Aykin [Ayk96].

Gärtner et al. [GMS05] extended shift design problem first time with breaks. However, the part of break placement are considered differently as break scheduling problem. There are fewer soft constraints, which are minimal and maximal length of break, minimal and

maximal distance of start of break from the shift begin, and minimal distance of end of break from the end of the shift. One or more breaks can be assigned to every shifts. The shifts are generated first based on local search, that we presented above ([GMS01] [MSS04]). The solution found by local search converted to shifts with one employee per day and greedy algorithm was used to assign breaks. The greedy algorithm finds the best position of the breaks depend on under- and over-covers. Next step is to update the length of breaks or shifts to increase or decrease to improve solution. Last step creates the solution with shifts and multiple employees. The algorithm found an average solution.

Gärtner et al. [GMS06] improved this algorithm by using integer programming formulation based on the set covering model of Dantzig [Dan54] for shift design phase, rather than local search. The shift design problem was defined with an extra component compared to the shift design problem presented in this master's thesis, that is the sum of the deviations of the shift lengths from the optimal shift length. Same steps were used for assigning breaks with the previous article [GMS05]. The method gave good results in practice.

Di Gaspero et al. [DGGM⁺10] [DGGM⁺13] proposed an innovative hybrid method that combines features of local search and constraint programming techniques. The problem is divided into two sub problems, where the local search technique is used to determine the shifts in the first phase and the constraint programming model to assign breaks. This approach could not improve the results obtained by solving the break scheduling problem separately after generating shifts.

CHAPTER 4

# Formulations

In this chapter, we will present our integer programming formulation for shift design and break scheduling problems in details. The integer programming approach is explained separately in two subsections. In each subsection, we describe the variables that we used, the constraints and the objective function.

## 4.1 Integer Linear Programming Model for Shift Design Problem

The explicit set covering formulation have been first proposed for the shift scheduling problem by Dantzig (1954) [Dan54]. Our integer programming formulation is also based on an explicit representation of shift design. To formulate the shift design problem explicitly, we generate all shifts including all feasible combinations of shift start times and lengths. The model uses the following variables.

### 4.1.1 Variables

The variables of integer programming formulation consist of two parts, input and decision variables.

**Input Variables**

We presented the original problem definition in the second chapter. Here, we will define the input and the generated variables from the given instance. For shift design problem formulation, the input variables are:

- *slotLength* : All time slots have the same length of interval *slotLength*. For the shift design problem usually the slot length is 15, 30 or 60 minutes.

- $daysPerCycle$ : The number of days in the planning period. $daysPerCycle$ is typically 7 days. The shift design problem has a cyclic structure. Assuming 7 days of period, the last time slot of the seventh day is equal to the first time slot of the first day.

- $n$ : Number of consecutive time slots with same $slotLength$. The calculation of $n$ variable is,

$$n = daysPerCycle * 24 * 60/slotLength \qquad (4.1)$$

- $m$ : Number of all enumerated possible shifts. All possible shifts are generated based on minimum / maximum shift start and minimum / maximum shift length. The variable $m$ is calculated as follows ($y$ is defined in the problem statement chapter as the number of shift types),

$$m = \sum_{i=1}^{y} DistinctStart_i * DistinctLength_i \qquad (4.2)$$

Suppose that, we have given the shift types in Table 2.1. Assuming a $slotLength$ of 60 minutes, morning shifts (M) can start at 05:00, 06:00, 07:00 or 08:00 and the length of shifts can be 7, 8 or 9 hours. The number of possible distinct shifts for the morning shift type is 12 (4 different starting hours * 3 different shift lengths). Further, there will be 9 possible shifts for the day shift, 9 for the afternoon shift and 9 for the night shift. In total, the number of possible shifts is 39, for the slotlength 60. If the slotlength is 30, there will be 110 possible shifts and if the slotlength is 15, there will be 360 possible shifts.

- $x_t$ : Number of employees required for time slot $t$ (defined as variable $w_i$ in the original problem definition).

$$x_t \geq 0 \quad \forall t = 1, 2, ..., n \qquad (4.3)$$

- $a_{ts}$ : For each possible shift $s$, we set the $a_{ts}$ variable to 1, if time slot $t$ is in the working period of shift $s$, 0 otherwise.

$$a_{ts} = \begin{cases} 1 & \text{if time slot t is a work period of shift s} \\ 0 & \text{otherwise} \end{cases} \qquad (4.4)$$

$$a_{ts} \in \{0, 1\} \quad \forall t = 1, 2, ..., n \quad \forall s = 1, 2, ..., m \qquad (4.5)$$

For instance, the first enumerated shift based on Table 2.1, will start at 05.00 and its length is 7 hours. All time slots between [05.00, 12.00) are set to 1 in each day of the planning period, other time slots must be equal to 0.

- $W_i$ : Weight of the three components in the objective function.

$$W_i \geq 0 \quad \forall i = 0, 1, 2 \tag{4.6}$$

## Decision Variables

The decision variables for the integer linear programming formulation for the shift design problem are given below,

- $b_s$ : For each enumerated shift, we use the variable $b_s$ indicating, whether shift $s$ is used or not.

$$b_s = \begin{cases} 1 & \text{if the shift } s \text{ is active} \\ 0 & \text{otherwise} \end{cases} \tag{4.7}$$

The shift $s$ is active, if at least one employee is assigned to it. The variable $b_s$ is initialized in Cplex Solver as,

```
1  b=cplex.numVarArray(m, 0, 1, IloNumVarType.Bool);
```

- $w_{sd}$ : Number of employees, that are assigned to a shift $s$ during the day $d$. The same shifts can have different number of employees in different days. This decision variable has two dimensions, shift number and day number. The initialization of the number of workers $w_{sd}$ in Cplex Solver is illustrated below,

```
1  w=new IloNumVar[m][];
2  for (int s = 0; s < m; s++){
3          w[s]=cplex.numVarArray(daysPerCycle, 0,
4                  Integer.MAX_VALUE, IloNumVarType.Int);
5  }
```

- The three components of the objective function are :

  $F_0$ : Sum of the excesses of workers in each time slot.

  $F_1$ : Sum of the shortages of workers in each time slot.

  $F_2$ : Number of shifts.

We initialized these three components of the objective function as follows in Cplex Solver,

```
1  F=cplex.numVarArray(3,0, Integer.MAX_VALUE, IloNumVarType.Int);
```

- $l_t$ : The load for time slot $t$. The load $l_t$ represents the sum of the number of workers, who works in time slot $t$ in all shifts. The initialization of load $l_t$ in Cplex Solver is,

19

```
1  l=cplex.numVarArray(n,0, Integer.MAX_VALUE, IloNumVarType.Int);
```

- $ex_t$ : The excesses of workers for each time slot $t$.

- $sh_t$ : The shortages of workers for each time slot $t$.

  We illustrate excesses $ex_t$ and shortages $sh_t$ of workers for each time slot in Cplex Solver below,

```
1  ex=cplex.numVarArray(n,0,Integer.MAX_VALUE,IloNumVarType.Int);
2  sh=cplex.numVarArray(n,0,Integer.MAX_VALUE,IloNumVarType.Int);
```

  We defined the variables, that we use in our integer programming model. In the next section, we will present the constraints.

### 4.1.2 Constraints

We will present the constraints, we use in our integer programming formulation in the next two sections based on the components of the objective function ( Excesses and shortages of workers in each time slot $(F_0, F_1)$ and number of shifts $(F_2)$).

**Excesses and Shortages of Workers in each Time Slot**

Usually in the shift scheduling problem, the aim is to minimize the number of employees, without any shortages. On the other hand, in shift design problem, we need to consider not only over staffing, but also under staffing component in the objective function. In order to find the sum of excesses and shortages of employees, first we need to define the load $l_t$ variable of time slot $t$ as follows,

$$l_t = \sum_{s=1}^{m}(a_{ts} * w_{s,\lfloor daysPerCycle*t/n \rfloor}) \quad \forall t = 1, 2...n \tag{4.8}$$

The load $l_t$ of each time slot $t$ is the sum of employees assigned to shifts that include this time slot. The variable $a_{ts}$ indicates, whether the time slot $t$ is in the time window of the shift $s$. If $a_{ts}$ variable is equal to 1, the number of workers $w_{sd}$ of the shift $s$ on this day is added to the equation. The day number $d$ for the variable $w_{sd}$ from the time slot $t$ is calculated with the following equation,

$$d = \lfloor t/(24 * 60/slotlength) \rfloor = \lfloor daysPerCycle * t/n \rfloor \tag{4.9}$$

There is one challenge to find the day number in $w_{sd}$ parameter from the time slot $t$. We need to consider the shifts, which start in one day $d$ and continuing into the following day $d + 1$. $w_{sd}$ is the number of workers of shift $s$, which starts to their shift in day $d$. From the calculation above, we needed to update the day number of the time slot of the next day in variable $w_{sd}$ to the one day before.

As an example, suppose we have 4 employees at one of the night shifts on the first day ($w_{s1} = 4$) and in the same shift we have 2 employees on the second day ($w_{s2} = 2$). If we calculate with the equation above the day value at 01.00 o'clock on the second day, the formula will find $w_{s2} = 2$, $w_{s2}$ is the number of workers on shift $s$ on the second day. However, we need to use the number of employees on the first day, because the shift started on Monday. Due to this problem, $nextDay_s$ variable is used that indicates, whether the shift continues next day or not. If a shift has $nextDay_s$ set to true and the time slot $t$ is after midnight, the day number will be decreased by one.

```
1   IloLinearNumExpr[] expr2 = new IloLinearNumExpr[n];
2   for ( int t = 0; t < n; t++){
3       expr2[t] = cplex.linearNumExpr();
4       for ( int s = 0; s < m; s++){
5           if(t % (24 * 60 / slotLength) < 12 * 60 / slotLength
6           && nextDay[s] == true ){
7               tempDay = ((daysPerCycle *t/n) + daysPerCycle −1)
8                           % daysPerCycle;
9           }
10          else
11              tempDay = (daysPerCycle *t/n);
12          expr2[t].addTerm(a[t][s], w[s][tempDay]);
13      }
14      cplex.addEq (l[t], expr2[t]);
15  }
```

From the formal definition of the shift design problem in [MSS04], the sum of excesses and shortages of workers is defined as follows,

$$F_0 = \sum_{t=1}^{n} \max(l_t - x_t, 0) * slotlength \tag{4.10}$$

$$F_1 = \sum_{t=1}^{n} \max(x_t - l_t, 0) * slotlength \tag{4.11}$$

Instead of using maximum operator in Cplex Solver, we calculate the excesses and shortages of workers in each time slot $t$ as follows,

$$ex_t \geq (l_t - x_t) * slotlength \quad \forall t = 1, 2...n \tag{4.12}$$

$$sh_t \geq (x_t - l_t) * slotlength \quad \forall t = 1, 2...n \tag{4.13}$$

where excesses $ex_t$ and shortages $sh_t$ variable are positive integers. If $l_t - x_t$ is negative, the value of parameter $ex_t$ is equal to 0 and if $l_t - x_t$ is positive, due to the minimization of excesses and shortages in objective function, the value of parameter $ex_t$ is equal to $l_t - x_t$. The shortage $sh_t$ can be also explained similarly.

The negative value of $x_t - l_t$ $(l_t - x_t)$, the value of the parameters $sh_t$ $(ex_t)$ is equal to 0 and for the positive value of $x_t - l_t$ $(l_t - x_t)$, due to the minimization of excesses and shortages in objective function, the equation is equal to the $sh_t = x_t - l_t$ $(ex_t = l_t - x_t)$

As we mentioned in Related Work, Gärtner et. al [GMS06] proposed an integer programming formulation for the shift design problem. They formulate this constraint as follows,

$$l_t * slotlength + sh_t - ex_t = x_t * slotlength \qquad (4.14)$$

This proposed constraint improved our solution, therefore we have also used this formulation.

This equation is illustrated in Cplex Solver as below,

```
1  IloLinearNumExpr[] expr3 = new IloLinearNumExpr[n];
2  for ( int t = 0; t < n; t++){
3      expr3[t] = cplex.linearNumExpr();
4      expr3[t].addTerm(slotLength, l[t]);
5      expr3[t].addTerm(-1.0, ex[t]);
6      expr3[t].addTerm(1.0, sh[t]);
7      cplex.addEq (expr3[t], slotLength*x[t]);
8  }
```

The calculation of the sum of excesses/ shortages of workers in each time slot:

$$F_0 = \sum_{t=1}^{n} ex_t \qquad (4.15)$$

$$F_1 = \sum_{t=1}^{n} sh_t \qquad (4.16)$$

```
1          cplex.addEq (F[0], cplex.sum(ex));
2          cplex.addEq (F[1], cplex.sum(sh));
```

**Number of Shifts**

One of the challanges with the shift design problem is that in order to minimize the number of shifts, we need to reuse the same shift on all days of the week and track the number of used shifts. Therefore, if any enumerated shift has an employee in at least one day ($\sum_{i=1}^{daysPerCycle} w_{sd} > 0$), the shift needs to be active ($b_s = 1$). Otherwise, if it is not active ($b_s = 0$), then the shift must not have any workers in any days ($\sum_{i=1}^{daysPerCycle} w_{sd} = 0$). The formulation is given below,

$$\sum_{d=1}^{daysPerCycle} w_{sd} \leq M * daysPerCycle * b_s \quad \forall s = 1, 2, ..., m \qquad (4.17)$$

22

We have used variable $M$ to get an upper bound of the maximal number of employees that each shift can have. The maximum number of workers each shift can have is the maximum number of needed employees over all time slots. Therefore,

$$M = \max x_t, \quad \forall t \in \{1, 2..., n\} \tag{4.18}$$

$M$ times *daysPerCycle* gives us the maximum bound of the sum of employees in each day. Instead of using the sum of workers in all days of every shift, we changed this constraint to,

$$w_{sd} \leq M * b_s \quad \forall s = 1, 2, ..., m \tag{4.19}$$

This new constraint is more efficient than the old one, due to the narrower feasible region. The illustration of the new version of the constraint in Cplex Solver is below,

```
1  IloLinearNumExpr[] expr1 = new IloLinearNumExpr[m];
2  for ( int s = 0; s < m; s++){
3      expr1[s] = cplex.linearNumExpr();
4      expr1[s].addTerm(b[s], instance.getMaxEmployeeNeeded());
5      for ( int d = 0; d < daysPerCycle; d++){
6          cplex.addLe(w[s][d], expr1[s]);
7      }
8  }
```

The calculation of the number of shifts:

$$F_2 = \sum_{s=1}^{n} b_s \tag{4.20}$$

```
1          cplex.addEq (F[2], cplex.sum(b));
```

### 4.1.3 Objective function

The objective function is combined with three weighted criteria.

$$\min \sum_{i=0}^{2} W_i * F_i \tag{4.21}$$

The minimization of objective function for the shift design problem is illustrated below,

```
1  IloLinearNumExpr obj = cplex.linearNumExpr();
2  for ( int i = 0; i < 3; i++){
3      obj.addTerm(weight[i], F[i]);
4  }
5  cplex.addMinimize(obj);
```

## 4.2 Integer Linear Programming Model for Break Scheduling Problem

In break scheduling problem, each duty of a shift in each day has several breaks and the *slotLength* is typically 5 minutes. The enumeration of all possible breaks including all feasible combinations of break start times and lengths ends up with large feasible set of integer linear formulation. It is almost impossible to solve this explicit formulation without any restriction. Nevertheless, due to the constraints based on work periods, we need to know the break location sequentially within their shift. Therefore, we give an integer programming model explicitly, but reduce all combinations of breaks.

The restrictions of our problem is given below,

- There are 7 soft constraints in the problem statement. Instead of using all of them as soft constraints in our formulation, we initialize our variables, such that they satisfy the first five soft constraints.

- We assume that every monitor break has duration of exactly 2 time slots (10 minutes).

- We assumed that the first three monitor breaks are before the lunch break and the remaining monitor breaks are after the lunch break.

We will present these restrictions in the further sections in more details. Our model uses the following variables.

### 4.2.1 Variables

The variables of integer programming formulation consist of two parts, input and decision variables.

**Input Variables**

In this section, we will present the given instance and will introduce new input variables, that are generated from the given variables. For break scheduling problem formulation, the input variables are:

- *slotLength* : The *slotLength* is usually 5 minutes for break scheduling problem.

- $n$ : Number of consecutive time slots with same slotlength. $n$ is calculated as follows,

$$n = 24 * 7 * 60 / slotLength \tag{4.22}$$

Based on formulation above and considering that *slotLength* is typically 5 minutes, number of consecutive time slots $n$ is equal to 2016.

- *sdd* : In break scheduling problem, we need to assign breaks for each duty of the shift in each day. Therefore, we use a variable *sdd* indicating the number of all shift-day-duty. Every shift has several workers in each day, to find the shift or the day from the shift-day-duty number, two tables are used. These tables *shiftSDD* / *daySDD* consist of shift-day-duty number and shift / day number.

- $s_i.breakTime$ : The duration of breaks in shift $s_i$ is calculated based on the length of a shift.

  if $shiftLength \leq 10$ hours,

  $$s_i.breakTime = \lfloor (minutes(ShiftLength) - 20)/50 \rfloor * 10 \qquad (4.23)$$

  else,

  $$s_i.breakTime = \lceil minutes(ShiftLength)/4 \rceil \qquad (4.24)$$

  In most cases, each shift-day-duty has one lunch break and several monitor breaks. Lunch breaks are 30 minutes long and the remaining break times are considered as monitor breaks.

  Suppose that we have a shift with a length of 8 hours. Based on 5 minutes *slotlength*, there are 96 time slots. Therefore, from the equation above, 18 time slots of breaks need to be assigned. 6 time slots are considered as a lunch break and the remaining 12 time slots are monitor breaks.

- $m$ : Number of breaks of shift $s_i$. As we mentioned before, we assume that each monitor break is 10 minutes (2 time slots). Therefore, the number of breaks is calculated with the equation below,

  $$s_i.m = (s_i.breakTime - s_i.lunchBreakTime)/2 + 1 \qquad (4.25)$$

- $r_t$ : Number of required staff for each time slot $t$. These numbers of employees are needed to be working at each time interval, an example of required employees for one day is shown in Figure 4.1.

  $$r_t \geq 0 \quad \forall t = 1, 2, ..., n \qquad (4.26)$$

- $shiftMinusRequirement_t$ : There are several shifts, that are characterized with start $s_i.start$ and length $s_i.length$. First, we need to convert each shift $s_i$ with a number of workers $s_i.w_j$ in each day $j$ to the number of people working in each time slot $t$. This working staff decreased by requirements of workers in each time slot $t$ to calculate $shiftMinusRequirement_t$ variables. In this calculation, we are not considering the breaks of workers. In Figure 4.2 is shown an example graph with required employees and present employees (shifts without breaks). The breaks will fill these space between two curves, as it is shown in Figure 4.3.

25
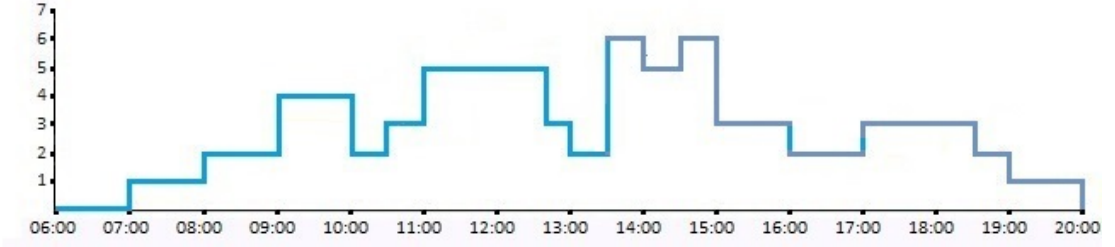
Figure 4.1: The required workers (Blue Line) over a day are shown
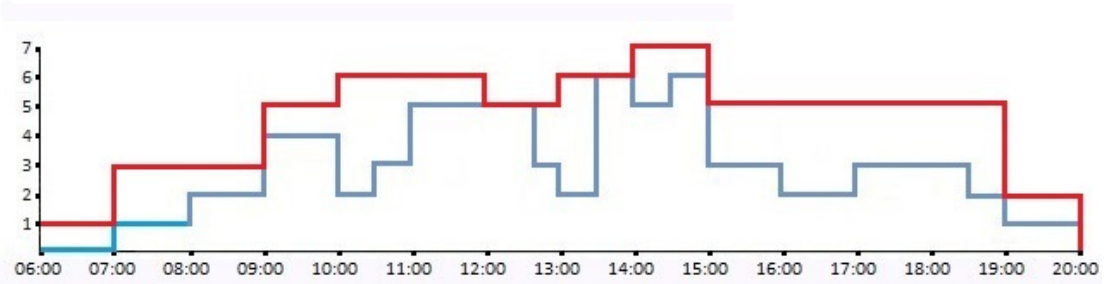


Figure 4.2: Shifts are scheduled without breaks. The required workers (Blue Line) and the present workers (Red Line) over a day are shown

$$shiftMinusRequirement_t \geq 0 \quad \forall t = 1, 2, ..., n \tag{4.27}$$

To calculate the number of workers in each time slot between these two curves, we use Algorithm 4.1.

In the Algorithm 4.1, first we set the $shiftMinusRequirement_t$ variable with the minus needed number of workers in each time slot $t$ (Line 2). The number of day cycle is considered 7 days (a week) for break scheduling problem (Line 5). Each day has $n/7$ time slots. Suppose that the *slotlength* is equal to 5, time slots [0-288) is the interval of the first day, second day is between [288, 576), .. , [1928, 0) is the interval of last day.

The night shifts of last day continue to the beginning of the week, due to the cyclic structure. Therefore, we need to apply $\mod n$. In line 7, the day number is converted to the first time slot of a day with $(n * j/7)(\mod n)$ and we add the equation $s_i.start + l$ to calculate the each time slot $t$ belongs to the interval of shift $s_i$ on day $j$. The right side of the equation is equal to the number of employees in day $j$ and shift $s_i$.

26
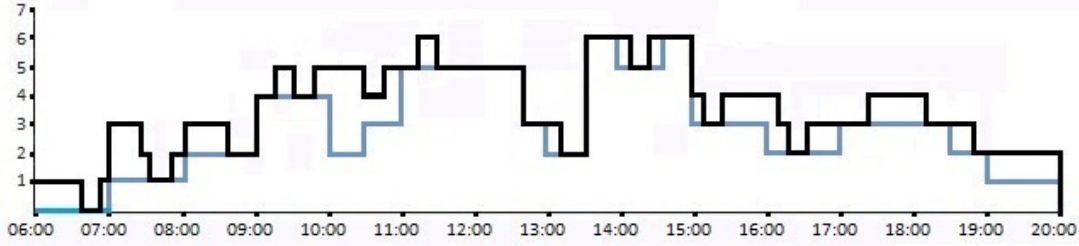
Figure 4.3: Breaks are assigned to each shift. The required employees (Blue Line) and the working employees (Black Line) over a day are shown

---

**Algorithm 4.1:** Convert Shift Schedules to the number of Assigned Workers in Each Time Slot

**Input**: $r_t$, $s_i.start$, $s_i.length$, $s_i.w_j$
**Output**: $shiftMinusRequirement_t$

1 **for** $t \leftarrow 0$ **to** *n-1* **do**
2     $shiftMinusRequirement_t = -r_t$
3 **end**
4 **for** $i \leftarrow 0$ **to** *Number of Shifts -1* **do**
5     **for** $j \leftarrow 0$ **to** *6* **do**
6        **for** $l \leftarrow 0$ **to** $s_i.length - 1$ **do**
7           $shiftMinusRequirement[(s_i.start + l + n * j/7)\%n]+ = s_i.w_j$ ;
8        **end**
9     **end**
10 **end**
11 **return** $shiftMinusRequirement_t$;

---

- As mentioned in [BGMS10], the common settings and the initialization of the used parameters in our formulation for the soft constraints $C0$ to $C6$ are given below (The weight of soft constraints are initialised with $W_i \quad \forall i = \{0, 1, 2...6\}$. ),

    - $C_0$ : The earliest start of the break (*earliestStart*) is half an hour after the beginning of the shift and the latest end (*latestEnd*) half an hour before the shift's end. The violation of this soft constraint is penalized with the weight $W_0$, which is 20.

    - $C_1$ : The earliest start of the lunch break (*lunchEarliestStart*) is $03 : 30$ hours and the latest end (*lunchLatestEnd*) $06 : 00$ hours after the beginning of the shift. The weight value $W_1$ of this constraint is equal to 10.

27

- $C_2$ : Work periods are the consecutive time slots between the breaks and also between the first break and start of a shift and the last break and end of a shift. These periods need to be between two parameters, that are the minimum length of work period $minWP$ and maximum length of work period $maxWP$. These parameters are set $00:30$ and $01:40$ and the weight value $W_2$ is equal to 20.

  Meanwhile, if the start time of the first break is before the earliest start and the end time of the last break is after the latest end, this situation violates two soft constraints, that are $C_0$ and $C_2$.

- $C_3$ : Employees need to have more break time $minBreakExceedsWorkLimit$, if they exceed a certain limit $workLimit$. This constraint is violated, if after 50 minutes of the work period, the employee have less than 20 minutes break. The weight $W_3$ is equal to 20.

- $C_4$ : The break duration of employees needs not to be less than 10 minutes ($minBreakLength$) and not more than one hour ($maxBreakLength$).The violation of break duration is penalized with the weight $W_4$, which is 1.

- $C_5$ : Sum of the shortages of employees in each time interval during the planning period. The weight value $W_5$ for each shortage is equal to 10.

- $C_6$ : Sum of the excesses of employees in each time interval during the planning period. The weight value $W_6$ for each excess is equal to 2 .

Beer et. al in [BGMS10] mentioned that in real world benchmarks, most of these constraints violations ($C_0$, $C_1$, $C_2$, $C_3$, $C_4$) are less than 5 percent for each instance, with respect to the weight values above. Therefore, we initialize our variables based on satisfying these first five constraints and evaluate the objective function with the remaining two soft constraints, that are excesses and shortages of workers in each time interval during the planning period.

**Decision Variables**

For the proposed integer linear programming formulation for break scheduling problem, the decision variables are given below,

**Initialization of Breaks**

Each shift-day-duty $s$ has several breaks. We initialize these breaks with three types of break variables, based on different time windows $TW_i$ ($i = \{0, 1, 2\}$ is the number of break types). These three break types are $bl_{sbt}$, $l_{st}$, $al_{sbt}$.

In break scheduling problem, breaks can be assigned into time slot $t$ between the earliest start of the break *earliestStart* and the latest end of the break *latestEnd*. Thus, $C_0$ constraint needs to be satisfied and the breaks must be into this interval. These three break type variables are assigned to true, if there exists a start of a break in the time slot $t$.

$$bl_{sbt} = \begin{cases} 1 & \text{if shift-day-duty } s \text{ has break } b \text{ before lunch, it starts from time slot } t \\ 0 & \text{otherwise} \end{cases}$$

(4.28)

$$l_{st} = \begin{cases} 1 & \text{if shift-day-duty } s \text{ has a lunch break, it starts from time slot } t \\ 0 & \text{otherwise} \end{cases}$$

(4.29)

$$al_{sbt} = \begin{cases} 1 & \text{if shift-day-duty } s \text{ has break } b \text{ after lunch, it starts from time slot } t \\ 0 & \text{otherwise} \end{cases}$$

(4.30)

These time slots are restricted with different time windows, based on types of breaks. We are assuming that every staff has three monitor breaks before the lunch break and $m - 4$ monitor breaks after the lunch break depending on $s_i.breakTime$ of shift $s_i$. We will present them as follows, and explain time windows in details:

- $bl_{sbt}$ : We assumed each shift-day-duty $s$ has three monitor breaks before lunch. We need to consider the followings to reduce the time window $TW_0$ (earliest and latest possible start) of this type of breaks,

  - The $bl_{sbt}$ can not start before the *earliestStart* (based on constraint $C_0$).

  - These breaks can not end after the *lunchLatestEnd* (based on constraint $C_1$). As an example with shift length 8 hours, the time window of before lunch break (Red) is illustrated below,
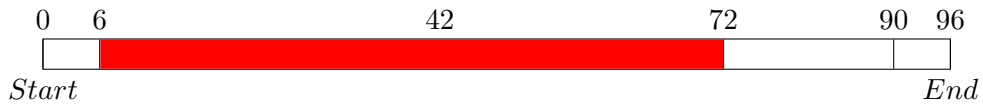


  - We can still restrict the time window $TW_0$. We need to consider that employees have a lunch break after $bl_{sbt}$ and before the *lunchLatestEnd*. The employees can have their lunch break at latest in the end of the lunch break period ($lunchLatestEnd - lunchBreakTime$) and before the lunch break, the staff must work minimum 6 time slots ($minWP$), due to the constraint $C_2$.

  - The monitor break variables have length of 2 time slots. Consider that time slot $t$ is the start time of the break. We need to decrease 2 more time slots. The

illustrated example of the restricted time window of before lunch break with shift length 8 hours is shown below (Red: Time Window of Before Lunch Break, Black: Monitor Break, Yellow: Minimum Working Period, Blue: Lunch Break),



From the restrictions above, the time window of $bl_{sbt}$ is calculated below,

$$es_0 = earliestStart \tag{4.31}$$

$$ls_0 = lunchLatestEnd - lunchBreakTime - minWP - 2 \tag{4.32}$$

The time windows of these three breaks are also tightened between each other from the following statements,

- The earliest start of the $bl_{s0t}$ (first before lunch break) is the *earliestStart*. However, it must end before the other remaining two before lunch breaks($bl_{s1t}, bl_{s2t}$). Therefore, we consider that $bl_{s2t}$ starts at $ls_0$. Before this break the employee needs to work $minWP$. $bl_{s1t}$ finishes at latest at this point of time. Therefore, if we decrease 2 time slots (length of second break) and $minWP$ (between the first and second break), we can find the latest point of the end of the first break. At last, we need to decrease 2 more time slot (length of first break) to find the latest start of the $bl_{s0t}$. The illustrated example of the restricted time window for the first before lunch break with shift length 8 hours is given below (Red : Time Window of Before Lunch Break, Black: Monitor Break, Yellow: Minimum Working Period, Blue: Lunch Break),
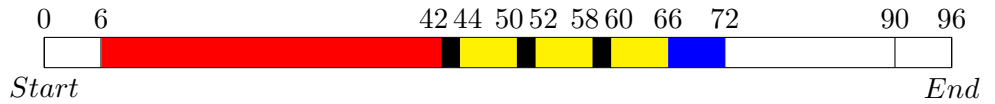


- We have already calculated the latest start of the $bl_{s1t}$, that is $ls_0 - minWP + 2$. The earliest start of the $bl_{s1t}$ must be after the $bl_{s0t}$, therefore, we need to add $minWP + 2$ to the to the earliest start of the $bl_{s0t}$. As an example of the restricted time window for the second before lunch break with shift length 8 hours is shown below (Red : Time Window of Before Lunch Break, Black : Monitor Break, Yellow : Minimum Working Period, Blue : Lunch Break),
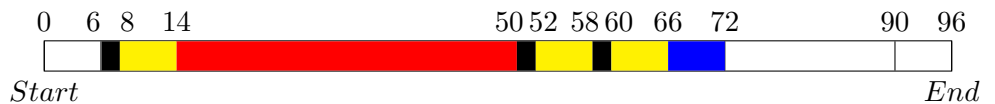
- With the same rule, the $bl_{s2t}$ must start $es_0 + 2*(minWP + 2)$ and latest start is the $ls_0$. The illustrated example of the restricted time window for the last before lunch break with shift length 8 hours is given below (Red : Time Window of Before Lunch Break, Black : Monitor Break, Yellow : Minimum Working Period, Blue : Lunch Break),



*Start*                                         *End*

As a summary, we added $i*(minWP+2)$ to the $es_0$ and decrease $(2-i)*(minWP+2)$ from the $ls_0$ where $i$ indicating the number of before lunch break -1. The updated interval of time window $TW_0$ of before lunch break is given below,

$$es_0 = earliestStart + i*(minWP + 2) \tag{4.33}$$

$$ls_0 = lunchLatestEnd - lunchBreakTime - (3 - i)*(minWP + 2) \tag{4.34}$$

$$\forall i = \{0, 1, 2\} \tag{4.35}$$

- $l_{st}$ : The shifts have at most one lunch break, therefore this variable is initialized with two dimensions shift-day-duty $s$ and time slot $t$. The lunch break can be between the $lunchEarliestStart$ and $lunchLatestEnd - lunchBreakTime$. This break has length 6 time slots. The illustration of time window $TW_1$ of lunch break with shift length 8 hours is shown below, (Blue: Time Window of Lunch Break, Black: Lunch Break),



*Start*                                         *End*

The earliest start and latest start of a lunch break is given below,

$$es_1 = lunchEarliestStart \tag{4.36}$$

$$ls_1 = lunchLatestEnd - lunchBreakTime \tag{4.37}$$

- $al_{sbt}$ : We assumed each shift-day-duty $s$ has $m - 4$ monitor breaks after lunch, based on length of the shift. We need to consider the followings to calculate the time window $TW_2$ of these types of breaks,

- The $al_{sbt}$ can not start before the *lunchEarliestStart* (based on constraint $C_1$).

- These breaks can not end after the *latestEnd* (based on constraint $C_0$). As an example with shift length 8 hours, the time window of after lunch break (Green) is illustrated below,
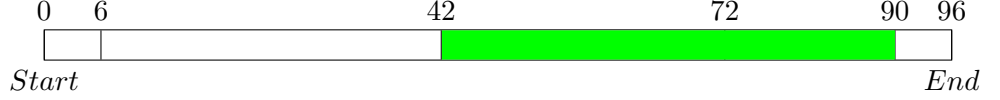


- Likewise, in before lunch break, we can also reduce the time window $TW_2$ of after lunch break. We need to consider that employees have a lunch break before these breaks and after the *lunchEarliestStart*. Therefore, assuming that the employees have lunch break in the beginning of the lunch break period (first 6 time slot) and after the lunch, the staff must have a working period minimum 6 time slots, due to the constraint $C_2$.

- The monitor break variables have a length of 2 time slots. Consider that time slot $t$ is the start time of the break. We need to decrease 2 time slots. The illustrated example of the restricted time window of after lunch break with shift length 8 hours is shown below (Green : Time Window of After Lunch Break, Black : Monitor Break, Yellow : Minimum Working Period, Blue : Lunch Break),
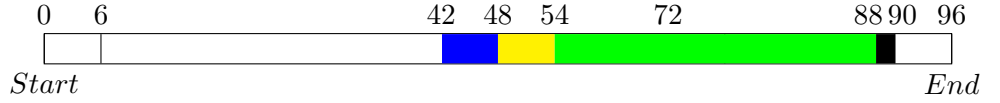


As we explained in before lunch breaks, we can also restrict the time window of each after lunch break $TW_2$ between each other. We add i * ( minWP +2 ) to earliest start of after lunch break ($lunchEarliestStart + lunchBreakTime + minWP$) and decrease (m - 5- i) * ( minWP +2 ) from latest start of after lunch break ($latestEnd - 2$) where $i$ indicating the number of after lunch break -1. From the all restrictions above, the earliest and latest start of after lunch breaks are calculated as given below,

$$es_2 = lunchEarliestStart + lunchBreakTime + minWP * (i + 1) + 2i \quad (4.38)$$

$$ls_2 = latestEnd - 2 - (m - 5 - i) * (2 + minWP) \quad (4.39)$$

$$\forall i = \{0, 1, .., m - 5\} \quad (4.40)$$

The illustrated example of the restricted time window for the first after lunch break with shift length 8 hours is given below (Green: Time Window of After Lunch Break, Black: Monitor Break, Yellow: Minimum Working Period, Blue: Lunch Break),

0   6                                      42   48   54              72 74 80 82 88 90   96

*Start*                                                                          *End*
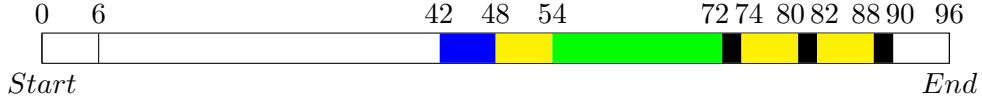
The illustrated example of the restricted time window for the second after lunch break with shift length 8 hours is shown below (Green: Time Window of After Lunch Break, Black: Monitor Break, Yellow: Minimum Working Period, Blue: Lunch Break),

0   6                                      42   48   54 56   62              80 82 88 90   96

*Start*                                                                          *End*
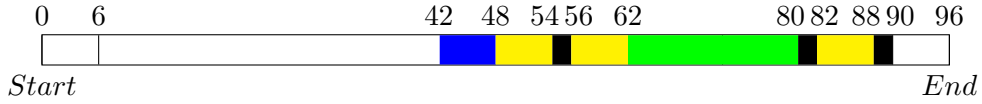
The illustrated example of the restricted time window for the last after lunch break with shift length 8 hours is given below (Green: Time Window of After Lunch Break, Black: Monitor Break, Yellow: Minimum Working Period, Blue: Lunch Break),

0   6                                      42   48   54 56 62 64   70              88 90   96

*Start*                                                                          *End*
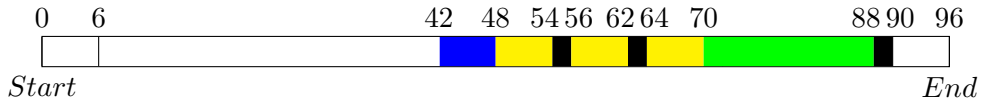
- $st_{sb}$ : Start time is an integer positive variable, represents the time slot of the start of the breaks within their shift. Every break $b$ of shift-day-duty $s$ have a start time and the value of start time is between this interval below,

$$0 \le st_{sb} \le ShiftLength - EarliestStart - LatestEnd - 2 \tag{4.41}$$

$$\forall s = \{0, 1...sdd - 1\} \quad \forall b = \{0, 1, ..., m - 1\} \tag{4.42}$$

We needed the start time of each break to calculate work period between each other. The equation to calculate the start time will be explained in constraint section.

- $wp_{sb}$ : There are two soft constraints based on a work period in the problem statement, however, as we mentioned before, we added this constraint directly to our initialization of the problem. $wp_{sb}$ is defined as follows,

  - The duration from the start time of the shift to the beginning of first break.
  - The duration from the end time of each break to the start time of the following break belongs to the same shift-day-duty.

– The duration from the end of the last break to the end of the shift.

We initialize the work period $wp_{sb}$ variable, based on start time $st_{sb}$ in the constraint section more clearly. This variable must be due to constraint $C_2$ between the interval [00:30, 01:40].

Based on the constraint $C_3$, if an employee exceeds 50 minute work period, he must have 20 minutes break. Each monitor break has 10 minutes length. Therefore, we initialize, that each work period has between 30 minutes to 50 minutes long, except two working periods. These two working periods are,

– The working period before the lunch break, because the lunch break is 30 minutes long, therefore the employee can exceed 50 minutes working period before.

– The working period after the last break. Because, the employee will finish his duty and will be free.

We initialize the each working period between interval,

$$00:30 \leq wp_{sb} \leq 01:40 \quad \forall s = \{0, 1...sdd - 1\} \quad \forall b = \{0, 1, ..., m\} \qquad (4.43)$$

We add the constraint whether working period is less or equal to 50 minutes or not in the following section.

- We added first five soft constraints into our formulation as an hard constraint. The remaining two components of the objective function are :

  – $C_5$ : Sum of excesses of employees in each time slot.
  – $C_6$ : Sum of shortages of employees in each time slot.

- To calculate these two components, we needed two variables $ex_t$, $sh_t$ like in shift design problem,

  – $ex_t$ : Excesses of workers in time slot $t$.
  – $sh_t$ : Shortages of workers in time slot $t$.

### 4.2.2 Constraints

We will present the constraints, we use in our integer programming formulation in 2 sections, soft and hard constraints.

**Hard Constraints**

- Each break of shift-day-duty $s$ must assign to exactly one time slot $t$. For each break type the equations are shown below,

  - For before lunch breaks $bl_{sbt}$,

$$\sum_{t \in TW_0} bl_{sbt} = 1 \quad \forall s = \{0, 1, ..., sdd - 1\} \quad \forall b = \{0, 1, 2\} \qquad (4.44)$$

  - For lunch breaks $l_{st}$,

$$\sum_{t \in TW_1} l_{st} = 1 \quad \forall s = \{0, 1, ..., sdd - 1\} \qquad (4.45)$$

  - For after lunch breaks $al_{sbt}$,

$$\sum_{t \in TW_2} al_{sbt} = 1 \quad \forall s = \{0, 1, ..., sdd - 1\} \quad \forall b = \{0, 1, ..., m - 4\} \qquad (4.46)$$

- $st_{sb}$ : Start time variable indicates the time difference between the start of a break and the belonging shift start $s_i.start$. The start times of each break are initialized based on each break type, are given below,

  - The dimension $t$ in variable $bl_{sbt}$, represent the start point of a break after the earliest start of before lunch breaks. We calculate the earliest start of the before lunch breaks $es_0$ as Earliest Start $+(minWP + 2) * i$, where $i$ is indicating the number of before lunch break. $st_{sb}$ is initialized for each before lunch break as follows,

$$st_{sb} = es_0 + t * bl_{sbt} \quad \forall s = \{0, 1...sdd - 1\} \quad \forall t \in TW_0 \quad \forall b = \{0, 1, 2\} \quad (4.47)$$

  - There is just one lunch break and it is the fourth break (b = 3). The earliest start of the lunch break $es_1$ is lunch earliest start. The initialization of it, is given below,

$$st_{s3} = es_1 + t * l_{st} \quad \forall s = \{0, 1...sdd - 1\} \quad \forall t \in TW_1 \qquad (4.48)$$

  - The earliest start of the after lunch breaks $es_2$ is initialized as Lunch Earliest Start + Lunch Break Time + $minWP + (minWP + 2) * i$ where $i$ is representing the number of after lunch break. The start times of first 4 breaks are initialized before, therefore we add 4 to the value of $b$ to initialize start time of after lunch breaks.

$$st_{s(b+4)} = es_2 + t * al_{sbt} \quad \forall s = \{0, 1...sdd-1\} \quad \forall t \in TW_2 \quad \forall b = \{0, 1, .., m-5\} \tag{4.49}$$

- $wp_{sb}$ : We add the constraint $C_2$ and $C_3$ into our formulation in this section. The end time of each break is the addition of start time and break duration (Monitor breaks : 2 time slots, Lunch breaks : 6 time slots). Therefore, for these constraints, the work period is initialized based on start time and duration of each break. The equations to calculate work periods are given below,

    – The first work period is equal to the start time of the first break, because the start time variable is the length from the shift start.

$$wp_{s0} = st_{s0} \quad \forall s = \{0, 1...sdd - 1\} \tag{4.50}$$

    – The last work period is equal to the equation given below,

$$wp_{sm} = ShiftLength - st_{s(m-1)} - 2 \quad \forall s = \{0, 1...sdd - 1\} \tag{4.51}$$

    – The work periods of the remaining breaks are between the end of the break and start of the following break. Therefore, they are initialized as follows,

$$wp_{sb} = st_{sb} - st_{s(b-1)} - 2 \quad \forall s = \{0, 1...sdd - 1\} \quad \forall b = \{1, 2.., m - 1\} \tag{4.52}$$

    The work period variable is already initialized between the interval [00:30, 01:40], due to constraint $C_2$. We need to decrease the length of the work period to $workLimit$ (50 minutes) based on constraint $C_3$, if the following break is shorter than 20 minutes ($minBreakExceedsWorkLimit$). The work period before lunch (b=2) and the last work period (b=m) can be more than 50 minutes. Therefore, we add constraints below,

$$wp_{sb} \leq workLimit \quad \forall s = \{0, 1...sdd - 1\} \quad \forall b = \{0, 1, 3, .., m - 1\} \tag{4.53}$$

**Soft Constraints**

As we mentioned before, the objective function consists of two remaining soft constraints ($C_5, C_6$) , these are excesses and shortages of employees in each time slot. To find these components, we use the $shiftMinusRequirement_t$ variable, that represent the number of employees remaining after we reduced the required employee of each time slot from the working employees (without considering breaks) belongs to all shifts.

In this part, we will include the breaks of employees and decrease from the variable $shiftMinusRequirement_t$ to the employees, that are at lunch or monitor breaks belongs to each shift-day-duty in each time slot. The result of this equation can be in each time slot,

- 0 : There are not any excesses or shortages of workers in time slot $t$.

- Positive : There are excesses of workers $ex_t$ in time slot $t$.

- Negative : There are shortages of workers $sh_t$ in time slot $t$.

The same as in shift design problem, we calculate the excesses and shortages of employees in time slot $t$ with the equation, is given below,

$$breaks_t - sh_t + ex_t = shiftMinusRequirement_t \quad \forall t = \{0, 1, 2, .., n-1\} \qquad (4.54)$$

where,

$breaks_t$ : Sum of employees have break in time slot $t$

$ex_t$ : This variable is a positive integer and the positive value of the equation $shiftMinusRequirement_t - breaks_t$ is equal to excesses $ex_t$ of employees in time slot $t$.

$sh_t$ : This variable is a positive integer and the negative value of the equation $shiftMinusRequirement_t - breaks_t$ is equal to shortages $sh_t$ of employees in time slot $t$.

We need to calculate the $breaks_t$ variable, that is the sum of all employees in before lunch, lunch or after lunch breaks in time slot $t$ in planning period (t =$\{0,1, ...,n\}$). The illustration of calculation $breaks_t$ variable and the excesses $ex_t$ and shortages $sh_t$ of employees in time slot $t$ is given in Algorithm 4.2. This variable is calculated based on 2 statements, which are given below,

- We initialized the before lunch breaks, lunch break or after lunch breaks based on time window of belonging shift-day-duty $s$ and time slot $t$ is the start time value of break between the time window of each break type. We need to convert this $t$ value to a general time slot in planning period (0,1, ...,n). For each break type the calculation is different, based on different time window.

- Each monitor break, before lunch breaks and after lunch breaks, is 2 time slots and a staff needs a full time slot to continue his work after his each break. This time slot is considered neither break, nor work period. However, we need to add these full time slot also to $breaks_t$ variable to calculate excesses and shortages of workers. In total monitor breaks need to be considered as 3 time slots and with this extra time slot, lunch breaks have 7 time slot length.

In Algorithm 4.2 , we use if statements (Lines 5, 9, 11) to get each type of break from the general time slot $i$ in planning period ($i = \{0, 1, ...n-1\}$) . Each calculation is different based on earliest start $es_0$ (Line 5), $es_1$ (Line 11), $es_2$ (Line 19) of break types. mod $n$ is used, due to the cyclic structure and the day number is converted to the first time slot of a day with $(daySDD.get(s) * n/7)(\mod n)$. In Line 6, 13, 20, we calculate the sum of employees of each break type based on duration of breaks.

**Algorithm 4.2:** Algorithm to calculate sum of all employees have break in time slot $t$ and find excesses and shortages of workers in time slot $t$

**Input**: $bl_{sbt}$, $l_{st}$, $al_{sbt}$, $s_i.start$, $shiftMinusRequirement_t$ ..

**Output**: $sh_t, ex_t$

**1 for** $i \leftarrow 0$ **to** $n - 1$ **do**

**2**     **for** $s \leftarrow 0$ **to** $sdd - 1$ **do**

**3**        **for** $t \in TW_0$ **do**

**4**           **for** $b \leftarrow 0$ **to** $2$ **do**

**5**              **if** $i = (t + s_i.start + earliestStart + (minWP + 2) * b + (daySDD.get(s) * n/7))$ mod $n$ **then**

**6**                 $breaks_i += bl_{sbt} + bl_{sb(t-1)} + bl_{sb(t-2)}$ ;

**7**              **end**

**8**           **end**

**9**        **end**

**10**        **for** $t \in TW_1$ **do**

**11**           **if** $i = (t + s_i.start + lunchEarliestStart + (daySDD.get(s) * n/7))$ mod $n$ **then**

**12**              **for** $j \leftarrow 0$ **to** $6$ **do**

**13**                 $breaks_i += l_{s(t-j)}$ ;

**14**              **end**

**15**           **end**

**16**        **end**

**17**        **for** $t \in TW_2$ **do**

**18**           **for** $b \leftarrow 0$ **to** $m - 5$ **do**

**19**              **if** $i = (t + s_i.start + lunchEarliestStart + 6 + minWP + (minWP + 2) * b + (daySDD.get(s) * n/7))$ mod $n$ **then**

**20**                 $breaks_i += al_{sbt} + al_{sb(t-1)} + al_{sb(t-2)}$ ;

**21**              **end**

**22**           **end**

**23**        **end**

**24**     **end**

**25**     $breaks_i - sh_i + ex_i = shiftMinusRequirement_i$ ;

**26 end**

**27 return** $sh_t, ex_t$;

Sum of excesses/ shortages of workers in each time slot is calculated,

- Sum of excesses of employees in each time slot:

$$C_5 = \sum_{t=0}^{n-1} ex_t \qquad (4.55)$$

- Sum of shortages of employees in each time slot:

$$C_6 = \sum_{t=0}^{n-1} sh_t \qquad (4.56)$$

### 4.2.3 Objective function

The objective function is combination of two remaining weighted criteria:

$$\min \sum_{i=5}^{6} W_i * C_i \qquad (4.57)$$

CHAPTER 5

# Computational Experiments

In this chapter, we present the computational results obtained by our integer linear formulations for shift design and break scheduling problems. We will explain the environment for both problems first and then we will give further details in two subsections. In each subsection, first different benchmark instances for both problems will be introduced and afterwards present the experiments with different ILP solvers. In the last stage the results will be compared with the best existing results for this problem.

The formulations presented were implemented using Java programming language in Eclipse. Furthermore, the simplex algorithm and branch and bound search are performed by the Cplex ILP Solver 12.6. Cplex Solver solve integer programming formulation efficiently. For the shift design problem, we also experimented with Gurobi Solver version 5.6.3 and its performance is compared with the Cplex Solver. We have used Cplex Solver with default parameters in the first step, however, in some instances, which take too long time, we experimented with some parameters regarding the termination of branch and bound.

All experiments were performed on a machine, 2.4GHz Intel Core i5 CPU, 8 GB of RAM

All results, that we obtained are validated. For shift design problem, we developed a validation program. This validator takes as an input the instance of shift design problem and the solution file, which is obtained by the integer programming model using Cplex Solver or Gurobi Solver. For break scheduling problem, we have used the validation program from the link below,

`http://www.dbai.tuwien.ac.at/proj/SoftNet/Supervision/Benchmarks/`
`Validator.zip`

This validator also takes as an input the instance of break scheduling problem and the solution file.

## 5.1 Shift Design

For the computational result of the shift design problem, we will explain first how these data sets are generated. In the second part, we will give our results and we will compare them with the the state-of-the-art solutions.

### 5.1.1 Description of the Instances

We have used the four different sets of problem instances for our experiments. The information about the data sets can be found in the link below, where the instances can be downloaded.

    http://www.dbai.tuwien.ac.at/proj/Rota/benchmarks.html

These first three benchmark sets are randomly generated and the data set 4 contains real life instance. These data sets have been first introduced in [MSS04] [Mus01]. These data sets have been considered also in the literature ([MSS04] [Abs13] [DGGK⁺07]). Below, we give a short description of these data sets based on [MSS04] [Mus01]

**Data Set 1**

The first data set is randomly generated. These instances have neither shortages, nor excesses of employees in each time slot. Therefore, the solution of shifts is generated based on shift start and shift length. For each shift type 1 to 5 shifts are generated randomly with equal probability. The slot length is chosen randomly to be 15, 30 or 60 minutes. The number of duties for each generated shift is chosen randomly between 1 to 5 duty. The week days have the same number of duties with 0.9 probability and for the weekend the probability of the value changes is 0.6. Each shift has the same number of duties on Saturday and Sunday with probability 0.9. Weights of shortages and excesses are 1 and shift weight is equal to the slot length (15, 30 or 60).

**Data Set 2**

The second set is generated in similar way like the first data set. There are not any over- or under-cover of workers. However, the first 10 instances (1-10) are generated based on 12 shifts, the second 10 instances (11-20) have 16 shifts and the remaining 10 instances (21-30) consists of 20 shifts.

**Data Set 3**

In the third set, problems are generated also from invalid shift, which can deviate up to 4 time slots from normal starting times and lengths. The same number of shifts (valid and invalid shifts) is used as in data set 2. For the problem in this data set, solution without excess and shortage does not exist.

42

**Data Set 4**

The last data set consist of three instances, the first one is a real life instance, taken from a call center. The second and third one is modified from the instance number 5 from set 3. The second instance has 30 minutes slot length instead of 60 minutes and the third one has doubled workforce.

## 5.1.2 Computational Results

For shift design problem, Cplex and Gurobi Solvers were used to solve integer linear programming formulations. The first and second data sets are generated without any under- or over-staffing. Therefore, to find optimal solutions is easier than for the third data set.

In Table 5.1 and Table 5.2, we present the optimal solution, number of shifts and the run time (in second) of Cplex and Gurobi Solver for the data set 1 and data set 2. The column previous result shows the best existing running times for these examples. We will discuss the comparison in the next section. We only run these two data sets once in both ilp solvers and except one instance (27.) in data set 2, we have obtained the optimum solutions, that have been already found in state-of-the-art [DGGK+07] [Abs13].

We found all the optimal solution within a short time for the first data set. Cplex Solver reaches these optimal solutions in shorter time than Gurobi Solver. Especially in instance 17, Gurobi Solver needs much longer time, although it also found all optimal solutions for data set 1.

As shown in the Table 5.2, for instance number 27 in data set 2, we stopped Cplex Solver after two hours. To solve this instance, Cplex parameters are used. We will present these parameters and modifications in the next section in details.

Cplex Solver shows mostly better performance compared to Gurobi Solver also in data set 2, however, in this data set Gurobi Solver has a shorter run time in several examples. Gurobi Solver could not solve three instances in this data set (23. - 27. - 28.) within 2 hours time limit.

The third data set is generated with invalid shifts. This can causes excesses and shortages of employees. The complexity of the problem is increased and to reach the optimal solution is harder. However, we have been able to find very good results. - For data set 3, we present our solution in two tables. The objective function value, over-staffing, under-staffing, number of shifts and the run time (in second) of Cplex Solver (Table 5.3) and Gurobi Solver (Table 5.4) are presented. We also run this data set once in both ILP solvers and for few instances the run times were over 2 hours. We consider changing Cplex parameters for these instances in the further section.

In third data set, Gurobi Solver could not obtain solution for 8 instances within 2 hours time limit and the run takes longer time than Cplex Solver. In several instances, Gurobi Solver finds different optimal solutions from Cplex Solver. That is, Gurobi Solver obtained not only fewer or more under- and over-staffing, but also fewer or more number of shifts, although the objective function values are the same.

| Instance | Optimal Solution | Nr. Of Shifts | Cplex (sec) | Gurobi (sec) | Previous Results (sec) | Resource |
|---|---|---|---|---|---|---|
| 1 | 480 | 8 | 0 | 0 | 0.07 | [DGGK$^+$07] |
| 2 | 300 | 10 | 14 | 75 | 16.41 | [DGGK$^+$07] |
| 3 | 600 | 10 | 0 | 1 | 0.11 | [DGGK$^+$07] |
| 4 | 450 | 15 | 14 | 169 | 26.75 | [Abs13] |
| 5 | 480 | 8 | 0 | 1 | 0.2 | [DGGK$^+$07] |
| 6 | 420 | 7 | 0 | 0 | 0.06 | [DGGK$^+$07] |
| 7 | 270 | 9 | 0 | 8 | 1.13 | [DGGK$^+$07] |
| 8 | 150 | 10 | 3 | 61 | 10.64 | [DGGK$^+$07] |
| 9 | 150 | 10 | 1 | 26 | 3.53 | [DGGK$^+$07] |
| 10 | 330 | 11 | 1 | 17 | 23.19 | [Abs13] |
| 11 | 30 | 2 | 1 | 1 | 0.21 | [DGGK$^+$07] |
| 12 | 90 | 6 | 1 | 4 | 0.25 | [DGGK$^+$07] |
| 13 | 105 | 7 | 1 | 13 | 0.35 | [DGGK$^+$07] |
| 14 | 195 | 13 | 56 | 318 | 60.97 | [DGGK$^+$07] |
| 15 | 180 | 3 | 0 | 0 | 0.04 | [DGGK$^+$07] |
| 16 | 225 | 15 | 65 | 258 | 151.78 | [DGGK$^+$07] |
| 17 | 540 | 18 | 29 | 1334 | 288.42 | [DGGK$^+$07] |
| 18 | 720 | 12 | 0 | 8 | 1.71 | [DGGK$^+$07] |
| 19 | 180 | 12 | 7 | 163 | 126 | [DGGK$^+$07] |
| 20 | 540 | 9 | 0 | 2 | 0.11 | [DGGK$^+$07] |
| 21 | 120 | 8 | 1 | 18 | 0.28 | [DGGK$^+$07] |
| 22 | 75 | 5 | 1 | 7 | 0.65 | [DGGK$^+$07] |
| 23 | 150 | 10 | 2 | 77 | 6.19 | [DGGK$^+$07] |
| 24 | 480 | 8 | 0 | 4 | 0.11 | [DGGK$^+$07] |
| 25 | 480 | 16 | 43 | 234 | 26.38 | [Abs13] |
| 26 | 600 | 10 | 1 | 27 | 1.5 | [DGGK$^+$07] |
| 27 | 480 | 8 | 0 | 2 | 0.07 | [DGGK$^+$07] |
| 28 | 270 | 9 | 0 | 14 | 2.24 | [DGGK$^+$07] |
| 29 | 360 | 12 | 1 | 27 | 10 | [DGGK$^+$07] |
| 30 | 75 | 5 | 1 | 5 | 0.26 | [DGGK$^+$07] |

Table 5.1: Times (in seconds) using Cplex, Gurobi Solver and shortest times in the literature to reach the best known solution for Data Set 1

| Instance | Optimal Solution | Nr. Of Shifts | Cplex (sec) | Gurobi (sec) |
|---|---|---|---|---|
| 1 | 720 | 12 | 4 | 7 |
| 2 | 720 | 12 | 3 | 4 |
| 3 | 360 | 12 | 11 | 19 |
| 4 | 360 | 12 | 4 | 7 |
| 5 | 720 | 12 | 4 | 6 |
| 6 | 360 | 12 | 3 | 4 |
| 7 | 720 | 12 | 1 | 3 |
| 8 | 180 | 12 | 406 | 1162 |
| 9 | 360 | 12 | 10 | 11 |
| 10 | 660 | 11 | 9 | 16 |
| 11 | 480 | 16 | 465 | 1947 |
| 12 | 900 | 15 | 41 | 9 |
| 13 | 900 | 15 | 29 | 5 |
| 14 | 840 | 14 | 9 | 3 |
| 15 | 480 | 16 | 318 | 528 |
| 16 | 240 | 16 | 36 | 24 |
| 17 | 960 | 16 | 24 | 2 |
| 18 | 840 | 14 | 28 | 14 |
| 19 | 240 | 16 | 90 | 59 |
| 20 | 960 | 16 | 11 | 1 |
| 21 | 600 | 20 | 101 | 79 |
| 22 | 1080 | 18 | 20 | 20 |
| 23 | 300 | 20 | 684 | > 7200 |
| 24 | 600 | 20 | 82 | 1300 |
| 25 | 600 | 20 | 39 | 118 |
| 26 | 1020 | 17 | 10 | 29 |
| 27 | | | | > 7200 |
| 28 | 300 | 20 | 424 | > 7200 |
| 29 | 1140 | 19 | 30 | 16 |
| 30 | 1020 | 17 | 28 | 53 |

Table 5.2: Times (in seconds) to reach the best known solution using Cplex and Gurobi Solver for Data Set 2

| Instance | Optimal Solution | Nr. Of Shifts | Nr. Of Undercover | Nr. Of Overcover | Time (sec) |
|---|---|---|---|---|---|
| 1 | 2385 | 13 | 825 | 1365 | 1 |
| 2 | 7590 | 17 | 2550 | 4530 | 33 |
| 3 | 9540 | 15 | 5280 | 3810 | 19 |
| 4 | 6540 | 15 | 960 | 5130 | 51 |
| 5 | 9720 | 11 | 3240 | 5820 | 4 |
| 6 | 2070 | 14 | 510 | 1350 | 8 |
| 7 | 6075 | 15 | 4755 | 1095 | 7 |
| 8 | 8580 | 13 | 3630 | 4560 | 1 |
| 9 | 6000 | 17 | 2850 | 2895 | 47 |
| 10 | 2940 | 18 | 1410 | 990 | 30 |
| 11 | 5190 | 16 | 2610 | 2100 | 199 |
| 12 | 4110 | 25 | 750 | 2985 | 143 |
| 13 | | | | | > 7200 |
| 14 | 9600 | 17 | 3000 | 5580 | 61 |
| 15 | 11250 | 18 | 4230 | 6480 | 166 |
| 16 | 10620 | 10 | 5580 | 4440 | 49 |
| 17 | 4680 | 19 | 2370 | 2025 | 210 |
| 18 | 6540 | 18 | 1500 | 4500 | 166 |
| 19 | | | | | > 7200 |
| 20 | 8910 | 18 | 7320 | 1050 | 21 |
| 21 | | | | | > 7200 |
| 22 | 12600 | 15 | 3900 | 8250 | 31 |
| 23 | 8280 | 15 | 4620 | 2760 | 7 |
| 24 | 10260 | 16 | 5760 | 3540 | 5 |
| 25 | 13020 | 15 | 6660 | 5460 | 4 |
| 26 | 12780 | 16 | 4770 | 7530 | 145 |
| 27 | 10020 | 16 | 4920 | 4140 | 3 |
| 28 | 10440 | 17 | 4020 | 5400 | 7 |
| 29 | 6510 | 19 | 4740 | 1200 | 295 |
| 30 | 13320 | 14 | 5040 | 7440 | 4 |

Table 5.3: Times (in seconds), objective function components and values using Cplex Solver for Data Set 3

| Instance | Optimal Solution | Nr. Of Shifts | Nr. Of Undercover | Nr. Of Overcover | Time (sec) |
|---|---|---|---|---|---|
| 1 | 2385 | 13 | 825 | 1365 | 5 |
| 2 | 7590 | 17 | 2400 | 4680 | 26 |
| 3 | 9540 | 15 | 5280 | 3810 | 23 |
| 4 | 6540 | 15 | 1260 | 4830 | 4698 |
| 5 | 9720 | 11 | 3420 | 5640 | 15 |
| 6 | 2070 | 14 | 510 | 1350 | 44 |
| 7 | 6075 | 15 | 4755 | 1095 | 29 |
| 8 | 8580 | 13 | 3630 | 4560 | 24 |
| 9 | 6000 | 18 | 2760 | 2970 | 128 |
| 10 | 2940 | 18 | 1470 | 930 | 48 |
| 11 | 5190 | 16 | 2610 | 2100 | 3731 |
| 12 | | | | | >7200 |
| 13 | | | | | >7200 |
| 14 | 9600 | 17 | 2880 | 5700 | 61 |
| 15 | 11250 | 18 | 4230 | 6480 | 770 |
| 16 | 10620 | 10 | 5400 | 4620 | 18 |
| 17 | 4680 | 19 | 2550 | 1845 | 1559 |
| 18 | 6540 | 18 | 1560 | 4440 | 851 |
| 19 | | | | | > 7200 |
| 20 | | | | | >7200 |
| 21 | | | | | > 7200 |
| 22 | | | | | >7200 |
| 23 | 8280 | 16 | 4800 | 2520 | 23 |
| 24 | 10260 | 16 | 5100 | 4200 | 5 |
| 25 | 13020 | 15 | 6840 | 5280 | 12 |
| 26 | | | | | > 7200 |
| 27 | 10020 | 15 | 5220 | 3900 | 8 |
| 28 | 10440 | 17 | 4860 | 4560 | 34 |
| 29 | | | | | > 7200 |
| 30 | 13320 | 14 | 4680 | 7800 | 14 |

Table 5.4: Times (in seconds), objective function components and values using Gurobi Solver for Data Set 3

| Instance | Optimal Solution | Nr. Of Shifts | Nr. Of Undercover | Nr. Of Overcover | Time (sec) |
|----------|------------------|---------------|-------------------|------------------|------------|
| 4-1 | 18420 | 12 | 10320 | 7740 | 1 |
| 4-2 | 9720 | 11 | 3000 | 6060 | 90 |
| 4-3 | 18780 | 11 | 7080 | 11040 | 18 |
| 3-5 | 9720 | 11 | 3240 | 5820 | 4 |

Table 5.5: Times (in seconds), objective function components and values using Cplex Solver for Data Set 4 and instance 5 of Data Set 3

| Instance | Optimal Solution | Nr. Of Shifts | Nr. Of Undercover | Nr. Of Overcover | Time (sec) |
|----------|------------------|---------------|-------------------|------------------|------------|
| 4-1 | 18420 | 12 | 10440 | 7620 | 1 |
| 4-2 | 9720 | 11 | 3720 | 5340 | 170 |
| 4-3 | 18780 | 11 | 7440 | 10680 | 28 |
| 3-5 | 9720 | 11 | 3420 | 5640 | 15 |

Table 5.6: Times (in seconds), objective function components and values using Gurobi Solver for Data Set 4 and instance 5 of Data Set 3

The result of the last data set is shown in Table 5.6. The second and third instances of dataset 4 are the modification of the instance 5 of dataset 3. Therefore, we added the result of it in the last row to see difference easier.

The running time of the real life instance (4-1) was one second and a solution with 12 feasible shifts was found. The second instance has half of the slot length of the 5. instance from the data set 3. As a result of this modification, the objective functions have the same value with different under- and over-staffing and solving this instance takes more time. The third one has doubled workforce, this modification ends up with a double objective function value, with also different excesses and shortages of workers and has also more run time.

Gurobi Solver obtains also an optimum solution in one second for the real life instance (4-1). The objective function of the second instance has the same value with different under- and over-staffing and take a lot more time also with Gurobi Solver. The third one ends up with a double objective function value, with also different excesses and shortages of workers.

In the next step, we will present the Cplex parameters and apply Cplex Solver again with different parameter values to have efficient solutions for several instances, which have running time more than two hours (27. Instance from Data set 2 and Instances 13-19-21 from Data set 3).

As a summary of the comparison between Cplex and Gurobi Solvers, Cplex Solver is faster than Gurobi Solver, both find the optimal values most of the time, however Gurobi Solver could not obtain optimal solutions for more instances due to two hours run time limit. Therefore, we will use Cplex Solver in the further runs.

### 5.1.3 Computational Results with some Limitation in Cplex Parameters

For our experimentations, we have experimented 3 different parameters in Cplex Solver. These parameters speed up the process, but change the optimality tolerance. One of these parameters is a time limit to terminate the Cplex Solver and the remaining two other parameters are related to gap value between the best found integer solution and optimal solution. These three parameters are defined in Cplex as follows,

- TiLim : This parameter is time limit in Cplex Solver, it can be set in seconds.

- EpGap : Relative optimality tolerance is set a certain percentage of gap to the optimal solution. This parameter can be between 0 and 1.

- EpAGap : Absolute optimality tolerance is set an absolute range of the optimal solution. This parameter can be any positive number.

If the objective function is a small number close to 0, then it is more reasonable to use absolute gap.

These parameters are set in Cplex Solver as follows,

```
cplex.setParam(IloCplex.DoubleParam.TiLim, 3600);
cplex.setParam(IloCplex.DoubleParam.EpGap, 0.1);
cplex.setParam(IloCplex.DoubleParam.EpAGap, 30);
```

The instances in data set 2 and data set 3, which running time is over the time limit, are run with different parameters in Cplex and the results are shown in Table 5.7. We have set the time limit parameter to half an hour and an hour and the EpAGap value to 30. We have tried different values for EpGap parameter for the instance in data set 2 and instances in data set 3. Because the solutions of data set 3 contains thousands of shortages and excesses of workers. Therefore, the objective functions are very high for these instances. The value of EpGap is related with the objective function value.

For instance, the optimum value of instance 27 in data set 2 is 300, according to [Abs13]. If we use the EpGap value equal to 2 %, the gap is set around 6 for optimum value around 300. As we know in data set 2, the objective function of the optimal solution is the cost of number shift. And the cost of each shift is 15 (for slot length = 15). Therefore, the gap of 6 between the founded integer value and optimal solution is too small.

The results of the instances with different parameters show that some of the parameters did not help us to find a solution, however, we have obtained good results also. We reached the best results for these instances by setting EpGap value to 0.01 (1% Gap) for data set 3 and to 0.1 (10% Gap) for data set 2. The larger gap value ends up with larger objective function value and the less gap does not change the result and increase the run time. Meanwhile, we have obtained the value 315 as a result of the instance 27 in data set 2, however, this result is not the best known solution considering to the article [Abs13].

| Instance | Paramater | Best Solution | Time (sec) | Nr. Of Shifts | Nr. Of Undercover | Nr. Of Overcover |
|----------|-----------|---------------|------------|---------------|-------------------|------------------|
| 2-27 | EpGap = 20 % | 330 | 473 | 22 | 0 | 0 |
| 2-27 | EpGap = 10 % | 315 | 655 | 21 | 0 | 0 |
| 2-27 | EpAGap = 5 % | - | > 7200 | - | - | - |
| 2-27 | EpAGap = 30 | - | > 7200 | - | - | - |
| 2-27 | TiLim = 1800 | 315 | 1800 | 21 | 0 | 0 |
| 2-27 | TiLim = 3600 | 315 | 3600 | 21 | 0 | 0 |
| 3-13 | EpGap = 2 % | 4620 | 120 | 26 | 2250 | 1980 |
| 3-13 | EpGap = 1 % | 4605 | 181 | 25 | 2370 | 1860 |
| 3-13 | EpGap = 0.5 % | 4605 | 866 | 25 | 2370 | 1860 |
| 3-13 | EpAGap = 30 | 4605 | 448 | 25 | 2370 | 1860 |
| 3-13 | TiLim = 1800 | 4605 | 1800 | 25 | 2370 | 1860 |
| 3-13 | TiLim = 3600 | 4605 | 3600 | 25 | 2370 | 1860 |
| 3-19 | EpGap = 2 % | 4905 | 92 | 24 | 3060 | 1485 |
| 3-19 | EpGap = 1 % | 4890 | 986 | 23 | 3645 | 900 |
| 3-19 | EpGap = 0.5 % | - | > 7200 | - | - | - |
| 3-19 | EpAGap = 30 | - | > 7200 | - | - | - |
| 3-19 | TiLim = 1800 | 4890 | 1800 | 23 | 3645 | 900 |
| 3-19 | TiLim = 3600 | 4890 | 3600 | 23 | 3645 | 900 |
| 3-21 | EpGap = 2 % | 5970 | 1046 | 32 | 2400 | 3090 |
| 3-21 | EpGap = 1 % | 5925 | 2026 | 29 | 2430 | 3060 |
| 3-21 | EpGap = 0.5 % | - | > 7200 | - | - | - |
| 3-21 | EpAGap = 30 | - | > 7200 | - | - | - |
| 3-21 | TiLim = 1800 | 5925 | 1800 | 29 | 2430 | 3060 |
| 3-21 | TiLim = 3600 | 5925 | 3600 | 29 | 2430 | 3060 |

Table 5.7: Times (in seconds), parameters, objective function components and values using different parameters in Cplex Solver for instances that exceed 2 hours time limit

In the next section, we compare our results with the existing state of the art results in the literature.

### 5.1.4 Comparison with Existing Results

In this section, we will compare our results with the previous best results, that were obtained by [DGGK+07], [MSS04], [Abs13]. Di Gaspero et al. published their results for the data set 1 and data set 3. Therefore, we will compare our results for two data sets.

First, we will present shortly these previous approaches and give the information about the machines, that were used by other approaches. In [MSS04] [DGGK+07] [Abs13] , the following approaches were proposed:

- LS : A local search solvers with a set of move types to explore the neighbourhood, is proposed by Musliu et al. [MSS04]. We note that algorithm proposed in this paper consider the optimization of an additional criteria regarding the avarage number of duties pro week. An extension of local search algorithm was proposed

and we compare in this paper with this algorithm, because in this paper also only the three criteria are considered.

- GrMCMF : A greedy construction heuristic based on min-cost max-flow is proposed by Di Gaspero et al. [DGGK+07].

- GrMCFC+LS : The hybrid solver is proposed by Di Gaspero et al. [DGGK+07]. The initial solution is constructed using new greedy heuristic based on min-cost max-flow. In the second stage, the local search paradigm is used to explore the neighbourhood.

- ASP : Different modelling approaches using answer set programming are proposed by Abseher [Abs13], but the results of [DGGK+07] could not be improved. Therefore, we compare with these approaches just in data set 1 to compare run time.

The local search approaches were implemented in C++ in the EASYLOCAL++ framework and GNU g++ compiler version 3.2.2 is used to compile. The experiments were performed on a machine, 1.5 GHz AMD Athlon PC running Linux kernel 2.4.21. The greedy constructed heuristic was coded in MS Visual Basic and runs on a MS Windows NT 4.0 computer. Approaches based on answer set programming are implemented on a machine Intel Xeon E5345 @ 2.33GHz 8 CPU-Cores and with a main memory 48 GB.

As we mentioned before, data set 1 consists of instances without under- or over-staffing. Therefore, the optimum results are achieved easily. We compare only our run times with the shortest needed time of each instance from the works of Di Gaspero et al. [DGGK+07] or Abseher [Abs13] to reach their best known solutions. These results are presented in Table 5.1.

Note that, because the experiments were performed with different machines. This comparison should be taken with caution and no clear conclusion can be drawn. However, from the Table 5.1 we can see that Cplex Solver solves problems in a short amount of time.

The comparison of the results for the third data set is shown in Table 5.8. We have achieved really good results with using Cplex Solver and found the optimal solution for almost all instances. For instances 13, 19, 21, the optimality could not be guaranteed, however these results were also better than the best known solutions. We have already presented our run time for data set 3 in Table 5.3 and for the instances 13, 19, 21 in Table 5.7.

| Instance | Cplex | GrMCMF | LS | GrMCMF+LS |
|---|---|---|---|---|
| 1 | **2,385** | 2,445.00 | 9,916.35 | 2,386.80 |
| 2 | **7,590** | 7,672.59 | 9,582.00 | 7,691.40 |
| 3 | **9,540** | 9,582.14 | 12,367.50 | 9,597.00 |
| 4 | **6,540** | 6,634.40 | 8,956.50 | 6,681.60 |
| 5 | **9,720** | 10,053.75 | 10,311.60 | 9,996.00 |
| 6 | **2,070** | 2,082.17 | 4,712.25 | 2,076.75 |
| 7 | **6,075** | **6,075.00** | 12,251.70 | 6,087.00 |
| 8 | **8,580** | 9,023.46 | 10,512.60 | 8,860.50 |
| 9 | **6,000** | 6,039.18 | 11,640.60 | 6,036.90 |
| 10 | **2,940** | 2,968.95 | 4,067.10 | 3,002.40 |
| 11 | **5,190** | 5,511.43 | 7,888.20 | 5,490.90 |
| 12 | **4,110** | 4,231.96 | 11,410.05 | 4,171.20 |
| 13 | **4,605** | 4,669.50 | 10,427.55 | 4,662.00 |
| 14 | **9,600** | 9,616.55 | 10,130.40 | 9,660.60 |
| 15 | **11,250** | 11,448.90 | 13,563.60 | 11,445.00 |
| 16 | **10,620** | 10,785.00 | 11,180.40 | 10,734.00 |
| 17 | **4,680** | 4,746.56 | 11,735.40 | 4,729.05 |
| 18 | **6,540** | 6,769.41 | 9,516.60 | 6,692.40 |
| 19 | **4,890** | 5,183.16 | 10,825.20 | 5,157.45 |
| 20 | **8,910** | 9,153.90 | 12,481.80 | 9,174.90 |
| 21 | **5,925** | 6,072.86 | 14,102.55 | 6,053.55 |
| 22 | **12,600** | 12,932.31 | 16,418.70 | 12,870.30 |
| 23 | **8,280** | 8,384.24 | 9,788.40 | 8,390.40 |
| 24 | **10,260** | 10,545.00 | 11,413.20 | 10,417.80 |
| 25 | **13,020** | 13,204.80 | 14,038.80 | 13,252.20 |
| 26 | **12,780** | 13,152.73 | 17,326.50 | 13,117.80 |
| 27 | **10,020** | 10,084.94 | 10,866.60 | 10,081.20 |
| 28 | **10,440** | 10,641.21 | 11,543.40 | 10,603.80 |
| 29 | **6,510** | 6,799.41 | 12,075.30 | 6,690.00 |
| 30 | **13,320** | 13,770.68 | 14,808.60 | 13,723.80 |

Table 5.8: The comparison of objective function values with literature of shift design problem for Data Set 3

## 5.2 Break Scheduling

We will explain first how data sets for break scheduling are generated. In the second part, we will give our results and we will compare them with the best solutions in the state-of-the-art.

### 5.2.1 Description of the Instances

We have used the two different sets of problem instances for our experiments. The information about the data sets can be found in the link below, where the instances can be downloaded.

    http://www.dbai.tuwien.ac.at/proj/SoftNet/Supervision/Benchmarks/

These instances are generated from the shift design problem. Afterwards, break time is computed for each shift, and breaks of 10, 20 minutes are added to the shifts, such that they satisfy the first five soft constraints of break scheduling problem ($C_0, C_1, C_2, C_3, C_4$). In the end, the needed numbers of employees in shift design problem instances are increased based on the number of needed employees, when breaks are included.

There are also real life instances, that have been used in previous work. These instances can not be solved with our formulation. The reason is that these instances have shifts with length more than 12 hours. As we mentioned before, we suppose that there will be 3 breaks before the lunch. For instance, a shift with a length of 12 hours should contain 36 time slots of 5 minutes for the breaks Considering 6 as lunch break and 3 times 10 minutes monitor breaks before lunch, we need to assign 12 breaks with 10 minute length after the lunch. Between these breaks, we need to consider the minimum working period of 30 minutes. It is impossible to assign 10 breaks after the lunch due to these conditions.

We will leave this part as a future work and compare our results only for the random example for break scheduling, that were used in previous works [WM14] [BGMS10].

### 5.2.2 Computational Results

We have reached the best solution with Cplex Solver for shift design problem. Therefore, we use only Cplex Solver to solve integer linear programming formulation for break scheduling problem. In Table 5.9, the results are shown for the instances, which have been also used in [WM14] [WM10]. The first five soft constraints ($C_0, C_1, C_2, C_3, C_4$) are always fullfilled and in the last two columns of the Table 5.9, we present the violationg over- and under-cover.

We have restricted our runs with two hours time limit. Due to our restrictions to our problem formulation, we can not guarantee optimality for these examples.

| Instance | Nr. Of Shifts(sdd) | Best Solution | Time (sec) | Nr. Of Undercover | Nr. Of Overcover |
|---|---|---|---|---|---|
| random1-1 | 137 | 84 | 3106 | 7 | 7 |
| random1-2 | 164 | 228 | 1874 | 19 | 19 |
| random1-5 | 141 | 360 | 580 | 30 | 30 |
| random1-7 | 157 | 228 | 7200 | 72 | 72 |
| random1-9 | 151 | 108 | 3654 | 9 | 9 |
| random1-13 | 124 | 348 | 1003 | 29 | 29 |
| random1-24 | 137 | 408 | 540 | 34 | 34 |
| random1-28 | 124 | 228 | 2349 | 19 | 19 |
| random2-1 | 179 | 636 | 7200 | 53 | 53 |
| random2-4 | 162 | 144 | 3633 | 12 | 12 |

Table 5.9: Times (in seconds), number of shifts, objective function components and values using Cplex Solver for randomly generated instance

| Instance | Nr. Of Shifts(sdd) | Cplex | MAPBS [WM14] | | |
|---|---|---|---|---|---|
| | | Best | Best | Avg | $\sigma$ |
| random1-1 | 137 | **84** | 346 | 440 | 48 |
| random1-2 | 164 | **228** | 370 | 476 | 65 |
| random1-5 | 141 | **360** | 378 | 418 | 29 |
| random1-7 | 157 | **408** | 496 | 583 | 42 |
| random1-9 | 151 | **108** | 318 | 423 | 51 |
| random1-13 | 124 | **348** | 370 | 445 | 55 |
| random1-24 | 137 | **408** | 542 | 611 | 43 |
| random1-28 | 124 | 228 | **222** | 318 | 71 |
| random2-1 | 179 | **636** | 724 | 889 | 75 |
| random2-4 | 162 | **144** | 476 | 535 | 45 |

Table 5.10: The comparison of our approach with the best existing results for randomly generated instance

### 5.2.3  Comparison with Existing Results

In this section, we will compare our results for each instance with the best existing result in the state of the art. Our formulation is not sufficient for the real life instance, therefore we will just compare with the generated random data sets for break scheduling problem.

Widl et al. proposed MAPBS approach in [WM14] [WM10] based on memetic algorithms and achieve the best known results for break scheduling problem, in each random generated instance.

The experiments in [WM14] [WM10] were performed on one core with 2.33Ghz of a QuadCore Intel Xeon 5345 with three runs being executed simultaneously, i.e. three cores being fully loaded. The machine provides 48GB of memory. Three runs were executed simultaneously. Each instance runs 10 times with a time limit 3046 seconds. To compare our results, we decrease the time limit of our runs also to 3046 seconds.

The comparison of the results is shown in Table 5.10. We obtain better solution for

most of the instances in randomly generated datasets. we could not improve the result for only the instance random1-28.

As a summary, integer programming formulation for the restricted break scheduling problem with Cplex Solver obtain the best known solutions for most randomly generated instances. However, we formulated this problem that assuming each shift has 3 breaks before the lunch and this assumption ends up with too many breaks after the lunch break for very long shifts. These assumptions must be reconsidered for the future works.

CHAPTER 6

# Conclusion and FutureWork

## 6.1 Summary

In this thesis, we developed an integer linear programming formulation for solving shift design and break scheduling problems. These problems are important employee scheduling problems and are introduced recently in the literature. The shift design problem arises in a variety of large organizations. It involves efficient usage of personnel resources to reduce costs as much as possible, while satisfying several constraints. Break scheduling problem is an important phase in the general employee scheduling in several organizations that needs a high level of concentration. The loss of concentration can end up with a dangerous consequences. Therefore, it is important that the workers have from time to time breaks to keep the concentration level high.

For the first problem shift design, we proposed integer linear programming model explicitly. This explicit model is generated based on enumeration of each shift from the possible shift starts and lengths. We applied state-of-the-art solvers Cplex and Gurobi to solve our integer linear programming for shift design problem. We have used the four different sets of problem instances for our experiments, including one real life instance. The Cplex Solver shows better time performance compared to Gurobi Solver. Therefore, we continued our further investigations with using Cplex. We obtained the optimal solution for 89 out of 93 instances from the literature. By experimentations with Cplex parameters, we could additionally obtain new results for three other instances. These Cplex parameters speed up the run process for hard instances, however, the optimality could not be guaranteed. For all instances except one (Instance 27 in dataset 2), we improved the best known result in literature.

For break scheduling problem, we proposed also explicit integer formulation, due to the constraints based on work periods between breaks. However, The enumeration of all possible breaks including all feasible combinations of break start times and lengths ends up with large feasible set of integer linear formulation. we give an integer programming model explicitly, but reduce all combinations of breaks. These new hard constraints are,

assuming each break is 10 minutes and the first three monitor breaks are before the lunch break. In addition to these constraints, we initialize the variables, such that they satisfy the first five constraints. We consider as objective function the remaining two soft constraints, sum of excesses and shortages of employees in each time slot.

We used only Cplex Solver to solve break scheduling problem. For this problem, there are some real life and two data sets consisting of randomly generated instances. The real life instances consist of assigned shifts with different shift lengths. We could not apply our approach to existing real life example from the literature with a longer shift lengths and larger number of breaks. However, we obtained the best existing results for randomly generated instances, except one instance.

## 6.2 Future Work

We improved some of the best known result for the shift design and break scheduling problem. However, we can reconsider and add some extensions and improvements to our formulations as future work:

- In the real life instances of break scheduling problem we could not obtain any solutions. Our integer programming formulation is not convenient for these instances, because, we formulated this problem with some restrictions and one of them is supposing that each shift contains exactly 3 breaks before the lunch. As a future work, we need to reconsider these assumptions and provide integer programming formulation for the general break scheduling problem.

- We proposed integer linear programming formulation for both problems separately and solved them sequentially. For the future work, it would be interesting to analyse solving both problem simultanuously.

- As mentioned by Di Gaspero et al. in [DGGM$^+$13], we can extend the problem by adding the new futures like employee qualification and task assignment. These extensions would be useful for professional planners to find better solutions in real life situations.

# Bibliography

[Abs13]     M. Abseher. Solving shift design problems with answer set programming. *Master's thesis, Vienna University of Technology, Vienna, Austria*, 2013.

[Ayk96]     T. Aykin. Optimal shift scheduling with multiple break windows. *Management Science, 42(4)*, pages 591 – 602, 1996.

[Ayk98]     T. Aykin. A composite branch and cut algorithm for optimal shift scheduling with multiple breaks and break windows. *Journal of the Operational Research Society, 49(6)*, pages 603 – 615, 1998.

[Ayk00]     T. Aykin. A comparative evaluation of modelling approaches to the labour shift scheduling problem. *European Journal of Operational Research, 125 (2)*, pages 381 – 397, 2000.

[BGM+08]    A. Beer, J. Gärtner, N. Musliu, W. Schafhauser, and W. Slany. Scheduling breaks in shift plans for call centers. *In Proceedings of The 7th International Conference on the Practice and Theory of Automated Timetabling, Montreal, Canada*, 2008.

[BGMS10]    A. Beer, J. Gärtner, N. Musliu, and W. Schafhauser. An AI-based break-scheduling system for supervisory personnel. *Intelligent Systems, IEEE 25 (2)*, pages 60 – 73, 2010.

[BJ90]      S. E. Bechtold and L. E. Jacobs. Implicit modeling of flexible break assignments in optimal shift scheduling. *Manage. Sci.*, 36(11):1339–1351, November 1990.

[Dan54]     G. B. Dantzig. A comment on Eddie's traffic delays at toll booths. *Operations Research, 2(3)*, pages 339 – 341, 1954.

[DGGK+07]   L. Di Gaspero, J. Gärtner, G. Kortsarz, N. Musliu, A. Schaerf, and W. Slany. The minimum shift design problem. *11th Annual European Symposium, Budapest, Hungary*, pages 593 – 604, 2007.

[DGGM+10]   L. Di Gaspero, J. Gärtner, N. Musliu, A. Schaerf, W. Schafhauser, and W. Slany. A hybrid LS-CP solver for the shifts and breaks design problem. *In Hybrid Metaheuristics, volume 6373 of Lecture Notes in Computer Science*, pages 46 – 61, 2010.

[DGGM$^+$13] L. Di Gaspero, J. Gärtner, N. Musliu, A. Schaerf, W. Schafhauser, and W. Slany. Automated shift design and break scheduling. *Automated Scheduling and Planning, volume 505 of Studies in Computational Intelligence*, pages 109 – 127, 2013.

[Edi54] L. C. Edie. Traffic delays at toll booths. *Journal Operations Research Society of America, 2(2)*, pages 107 – 138, 1954.

[GMS01] J. Gärtner, N Musliu, and W Slany. Rota: a research project on algorithms for workforce scheduling and shift design optimization. *AI Commun 14 (2)*, pages 83 – 92, 2001.

[GMS05] J. Gärtner, N. Musliu, and W. Slany. A heuristic based system for generation of shifts with breaks. *Applications and Innovations in Intelligent Systems XII*, pages 95–106, 2005.

[GMS06] J. Gärtner, N. Musliu, and W. Slany. Comparing mathematical and local search heuristic for solving the minimum shift design problem with breaks. 2006.

[MMT00] A. Mehrotra, K. E. Murphy, and M. A. Trick. Optimal shift scheduling: A branch-and-price approach. *Naval Research Logistics (NRL), 47 (3)*, pages 185 – 200, 2000.

[Moo76] S. L. Moondra. An L.P. model for workforce scheduling in banks. *Journal of Bank Research, 6*, pages 299 – 301, 1976.

[MSS04] N. Musliu, A. Schaerf, and W. Slany. Local search for shift design. *European Journal of Operational Research, 153(1)*, pages 51 – 64, 2004.

[MSW09] N. Musliu, W. Schafhauser, and M Widl. A memetic algorithm for a break scheduling problem. *In: 8th Metaheuristic International Conference, Hamburg, Germany*, 2009.

[Mus01] N. Musliu. Intelligent search methods for workforce scheduling: New ideas and practical applications. *PhD thesis, Vienna University of Technology*, 2001.

[RCS10] M. Rekik, J. F. Cordeau, and F. Soumis. Implicit shift scheduling with multiple breaks and work stretch duration restrictions. *Journal of Scheduling*, 13(1):49–75, 2010.

[Tho95] G. M. Thompson. Improved implicit optimal modeling of the labor shift scheduling problem. *Manage. Sci.*, 41(4):595–607, April 1995.

[Wid10] M. Widl. Memetic algorithms for break scheduling. *Master's thesis, Vienna University of Technology, Vienna, Austria*, 2010.

60

[WM10]     M. Widl and N. Musliu. An improved memetic algorithm for break schedul-
           ing. *HM'10 Proceedings of the 7th international conference on Hybrid
           metaheuristics*, pages 133 – 147, 2010.

[WM14]     M. Widl and N. Musliu. The break scheduling problem: complexity results
           and practical algorithms. *Memetic Computing, 6 (2)*, pages 97 – 112, 2014.