FAKULTÄT
FÜR !NFORMATIK
Faculty of Informatics

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

Wien, 23. April 2015

Deniz Kocabas

Abstract

Shift design and break scheduling problems that belong to class of NP-hard problems are variants of shift scheduling problem and are introduced recently in the literature. 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.

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. 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 slots during the planning period. And in second part we assign the breaks within their shifts conveniently respect to several constraints and also keep the deviation of workforce for the timeslots as minimal as possible. 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. For break scheduling, due to the large number of constraints and variables, we restricted the problem definition.

The simplex algorithm and branch and bound search are performed by the Cplex and Gurobi Solver, to solve the integer programming model for the data sets of shift design problem. The Cplex Solver shows faster performance compared to Gurobi Solver. Therefore, we continue our experiments using only Cplex Solver. We tried different parameters in Cplex Solver, with the instances exceed 2 hours time limit for the shift design problem. These parameters speed up the process, meanwhile change the optimality tolerance.

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 our restrictions in this problem. We improved the previous solutions and obtained the best known results in the state-of-the-art for both shift design and break scheduling problem.

Acknowledgements

Foremost, I would like to express my sincere gratitude to my supervisor Priv.-Doz. Dr. Nysret Musliu for the continuous support of my master thesis, for his patience, motivation, enthusiasm, and immense knowledge. His guidance and constructive critism helped me in all the time of research and writing of this thesis. I could not have imagined having a better advisor and mentor for my master thesis.

Besides my advisor, I would also like to thank the rest of the people of the Department for their help and support.

Last but not the least, I would like to thank my family, especially to my father Mehmet Kocabas and my mother Ummuhan Kocabas, for supporting me throughout my life.

Contents

List of Figures

List of Tables

List of Algorithms

CHAPTER 1

Introduction

1.1 Motivation

In this thesis, we investigated an integer programming approach to solve two NP-hard problems, shift design and break scheduling. These two problems are variants of the shift scheduling problem 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. Solving this shift scheduling problem was originally proposed by Dantzig [Dan54] by the set covering formulation. 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. The motivations of the shift design and the break scheduling problems are presented below separately in two sections.

1.1.1 Shift Design

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.

1.1.2 Break Scheduling

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 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.

1.2 Aim of The Work

The aim of our thesis consists of four main parts:

- The integer programming formulation will be proposed for shift design and break scheduling problems. We plan to study different formulations for both problems.
- To solve the models, we will apply the Cplex Solver and Gurobi Solver. We will compare experimentally the two solvers on benchmark instances for shift design problem from the literature.
- For experimentations, different parameters in Cplex Solver will be used. These parameters speed up the process, meanwhile change the optimality tolerance.
- To measure the performance of an integer programming approach, the results will be compared with the existing state-of-the-art's results in the literature.

1.3 Results of the Master's Thesis

We obtained the results of this master 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.

- Cplex Solver obtains the results in less time compared to Gurobi Solver for datasets of the 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 three different parameters of Cplex Solver are performed. By using these parameters, we can not guarantee the optimal solution is found. However, these long lasting instances by using these parameters also achieved better solutions compared to previous results.
- 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 our restrictions in this problem.

1.4 Structure of the Master's Thesis

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

- In **Chapter 2**, we will 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 will give an overview of state-of-the-art. We will discuss the proposed integer linear programming formulations for a similar problem called shift scheduling problem and then we will give the related work for shift design and break scheduling problems.
- In **Chapter 4**, we will present our integer linear programming formulations for both problems in details. The variables, constraints and objective functions will be described.
- In **Chapter 5**, we will present the computational results obtained by our integer linear programming formulations for shift design and break scheduling problems. We will compare the two integer linear programming solvers, Cplex Solver and Gurobi Solver for the datasets of shift design problem. We will experiment different parameters in Cplex Solver for long lasting instances of shift design problem. At last, we will 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.

CHAPTER 2

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.

As we mentioned above, we avoided some of the instances, that are necessary to calculate the distance of the average number of duties per week per employee. To disregard this fourth component of the objective function is reducing the complexity of the problem statement. This criterion is also excluded in [DGGK+07].

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,

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

  otherwise,

$$s_i.breakTime = ceil(minutes(ShiftLength)/4) \tag{2.2}$$

- 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.

$$s_i.start \leq b_j.start \leq b_j.end \leq s_i.end \tag{2.3}$$

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

$$b_j.start \leq b_j.end \leq b_k.start \leq b_k.end \tag{2.4}$$

  or

$$b_k.start \leq b_k.end \leq b_j.start \leq b_j.end \tag{2.5}$$

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

$$\sum_{b_j \in s_i} b_j.duration = s_i.breakTime \tag{2.6}$$

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:

$$b_j.start \geq s_i.start + distanceToShiftStart \tag{2.7}$$

$$b_j.end \leq s_i.end + distanceToShiftEnd \tag{2.8}$$

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,

$$b_{lb}.start \geq s_i.start + distanceToShiftStartLB \tag{2.9}$$

$$b_{lb}.end \leq s_i.end + distanceToShiftEndLB \tag{2.10}$$

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:

$$minWorkDuration \leq b_1.start - s_i.start \leq maxWorkDuration \tag{2.11}$$

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

$$minWorkDuration \leq s_i.end - b_m.end \leq maxWorkDuration \tag{2.13}$$

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 ($minTsCount$):

$$b_1.start - s_i.start \geq workLimit \Rightarrow b_1.duration \geq minTsCount \tag{2.14}$$

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

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 intervals, $[a_t, a_{t+1})$. Sum of the shortages of workers in each time slot during the planning period indicating the $C6$.

C_7: Excess of Employees. At most w_t employees should be working in each time intervals $[a_t, a_{t+1})$. Sum of the excesses of workers in each time slot during the planning period indicating the $C7$.

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, with minimizing the soft constraints.

CHAPTER 3

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 design 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 these first publications; Musliu et al. proposed a local search approach 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 to consider every change of the requirements. 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, that presented in the previous chapter 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. Tabu search mechanism and simulated annealing algorithm [BGMS10] are implemented 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. 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.

Widl [WM14] [Wid10] [WM10] proposed 2 new memetic algorithms. 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 search is used to prevent re-visiting old computed solutions in local search. To improve the first algorithm, 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

Real life shift design and break scheduling problem arise in different areas of industry. 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. The break scheduling definition is described differently as we presented in Chapter 2 before. 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 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, 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* and it is usually 15, 30 or 60 minutes for the shift design problem.

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	13:00	15:00	07:00	09:00
N	21:00	23:00	07:00	09:00

Table 4.1: An example of shift types

- *daysPerCycle* : The number of days in the planning period. *daysPerCycle* is considered typically a week (7 days). The shift design problem has a cyclic structure. Assuming 7 days of period, the last time slot t_{n+1} of the seventh day is equal to the first time slot t_1 of the first day.

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

$$n = daysPerCycle * 24 * 60/slotLength \tag{4.1}$$

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

$$m = \sum_{i=1}^{y} DistinctStart_i * DistinctLength_i \tag{4.2}$$

  Suppose that, we have given the shift types in Table 4.1. Assuming a *slotLength* of 60 minutes, morning shifts can start 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, 9 for the afternoon and 9 for the night shifts. In total, the number of possible shifts including all feasible combinations of shift start times and lengths is 39, for the slotlength 60. 110 possible shifts for the slotlength 30 and 360 possible shifts for the *slotLength* 15.

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

$$x_t \geq 0 \quad \forall t = 1, 2, ..., n \tag{4.3}$$

- 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.

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

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

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 t between [05.00, 12.00) set to 1 in each day of the time period, likewise [12.00-05.00) must be equal to 0.

- W_i : Weight of the three component.

$$W_i \geq 0 \quad \forall i = 1,2,3 \tag{4.6}$$

Decision Variables

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

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

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

The variable b_s is initialized in Cplex Solver as,

```
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 integer programming variable has two dimensions, shift number and day number. The initialization of the number of workers w_{sd} in Cplex Solver is illustrated below,

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

- 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,

```
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,

```
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,

```
ex=cplex.numVarArray(n,0,Integer.MAX_VALUE,IloNumVarType.Int);
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 2 sections based on the component 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 (F_0, F_1)

The shift scheduling problem focuses on minimum 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,

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

The load l_t of each time slot t is the sum of employees for every possible shift work during this period. The variable a_{ts} is used, 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,

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

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 shift on the first day ($w_{s1} = 4$) and in the same shift have 2 employees on the second day ($w_{s2} = 2$). If we calculate with the formulation 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, whether the shift continues in time slot of the next day or not. In every shift, that has true value of $nextDay_s$ variable, then if the time slot t is after midnight, the day number will be decreased one.

```
IloLinearNumExpr[] expr2 = new IloLinearNumExpr[n];
for ( int t = 0; t < n; t++){
    expr2[t] = cplex.linearNumExpr();
    for ( int s = 0; s < m; s++){
        if(t % (24 * 60 / slotLength) < 12 * 60 / slotLength
        && nextDay[s] == true ){
            tempDay = ((daysPerCycle *t/n) + daysPerCycle -1)
                        % daysPerCycle;
        }
        else
            tempDay = (daysPerCycle *t/n);
        expr2[t].addTerm(a[t][s], w[s][tempDay]);
    }
    cplex.addEq (l[t], expr2[t]);
}
```

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,

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

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

where excesses ex_t and shortages sh_t variable are positive integers. 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,

$$l_t * slotlength + sh_t - ex_t = x_t * slotlength \tag{4.14}$$

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

This equation is illustrated in Cplex Solver as below,

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

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

$$F_0 = \sum_{t=1}^{n} ex_t \tag{4.15}$$

$$F_1 = \sum_{t=1}^{n} sh_t \tag{4.16}$$

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

Number of Shifts (F_2)

One of the challanges with the shift design problem is in order to minimize the number of shifts, we need to reuse the same shifts 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 shift must not have any workers in any days ($\sum_{i=1}^{daysPerCycle} w_{sd} = 0$). The formulation is given below,

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

We have used variable M to get an upper bound of the number of employees each shift can maximum have. The maximum number of workers each shift can have is the maximum number of needed employees in 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,

```
IloLinearNumExpr[] expr1 = new IloLinearNumExpr[m];
for ( int s = 0; s < m; s++){
    expr1[s] = cplex.linearNumExpr();
    expr1[s].addTerm(b[s], instance.getMaxEmployeeNeeded());
    for ( int d = 0; d < daysPerCycle; d++){
        cplex.addLe(w[s][d], expr1[s]);
    }
}
```

The calculation of the number of shifts:

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

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

4.1.3 Objective function

The objective function is combined with three weighted criteria.

$$\min \sum_{i=1}^{3} W_i * F_i \tag{4.21}$$

To create a minimization objective function for shift design problem and adding to IloCplex in Cplex Solver,

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

4.2 Integer Linear Programming Model for Break Scheduling Problem

Due to the large amount of variables and constraints, we had difficulties formulate an integer linear programming model for break scheduling problem. We considered several different formulations, however, we achieved to solve this problem with only one of these formulations, after some restrictions to the problem statement. Except this formulation, Cplex Solver needed too much running time and we terminated after several hours. We will only present this proposed formulation.

In break scheduling problem, each duty of a shift in each day has several breaks and the *slotLength* is typically 5 minutes. To enumerate all possible breaks including all feasible combinations of break start times and lengths end 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 period, we need to know the break location sequentially within their shift. Therefore, we formulated integer problem model explicitly with reducing all feasible combinations of breaks.

The restrictions of our problem to reduce variables and constraints are,

- There are 7 soft constraints in the problem statement. Instead of using all of them as soft constraints, we initialize our variables, that satisfy these first five soft constraints, except the sum of excesses and shortages of workers in each time slot, in our formulation.
- We assumed every monitor breaks have duration exactly 2 time slots (10 minutes) not more or less.
- We assumed that the first three monitor breaks are before the lunch break and the remaining monitor breaks assign after the lunch.

We will present these restrictions in the further explanations in more details. The 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. The calculation of n variable is,

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

  Based on formulation above and considering *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 in each day belonging to all shifts. 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 hash tables are used. These hash tables $shiftSDD$ / $daySDD$ consist of shift-day-duty number and shift / day number.
- $s_i.breakTime$: The duration of breaks of shift s_i is calculated, based on the length of a shift.

  if $shiftLength \leq 10$ hours,

  $$s_i.breakTime = floor((minutes(ShiftLength) - 20)/50) * 10 \tag{4.23}$$

  else,

  $$s_i.breakTime = ceil(minutes(ShiftLength)/4) \tag{4.24}$$

  In most cases, each shift-day-duty have 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 8 hours. Based on 5 minutes *slotlength*, there are 96 time slots. Therefore, from the equation above, 18 time slots of breaks needs to be assigned. 6 time slots are considered as a lunch break and the remaining 12 time slots are for the monitor breaks.

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

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

- 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.

$$r_t \geq 0 \quad \forall t = 1, 2, ..., n \tag{4.26}$$

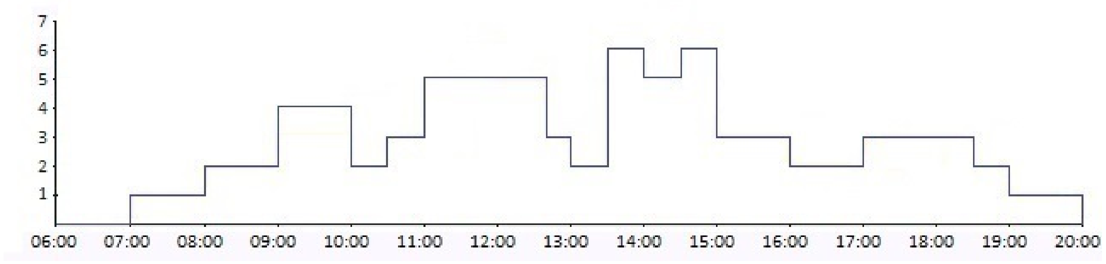


Figure 4.1: The needed workers over planning period is shown with the blue line.

- $shiftMinusRequirement_t$: There are several shifts, that are characterized with start $s_i.start$ and length $s_i.length$ of shift s_i. 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. We need to calculate this working staff decreased by requirements of workers $shiftMinusRequirement_t$ in each time slot t . 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.

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

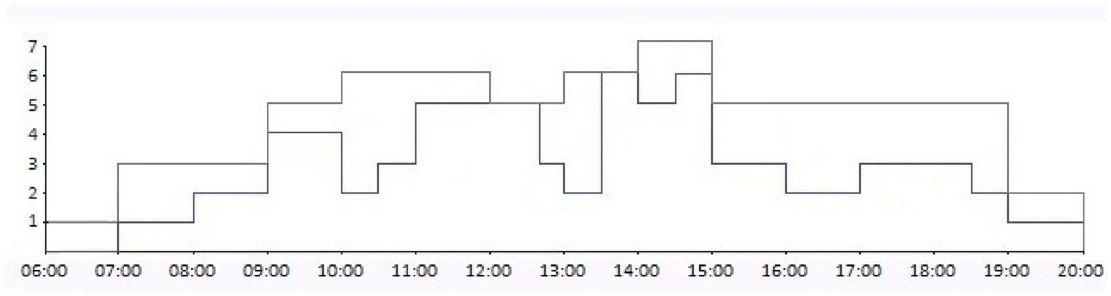


Figure 4.2: Shifts are scheduled without breaks, The required workers (Blue Line) and present workers (Gray Line) are shown over planning period.

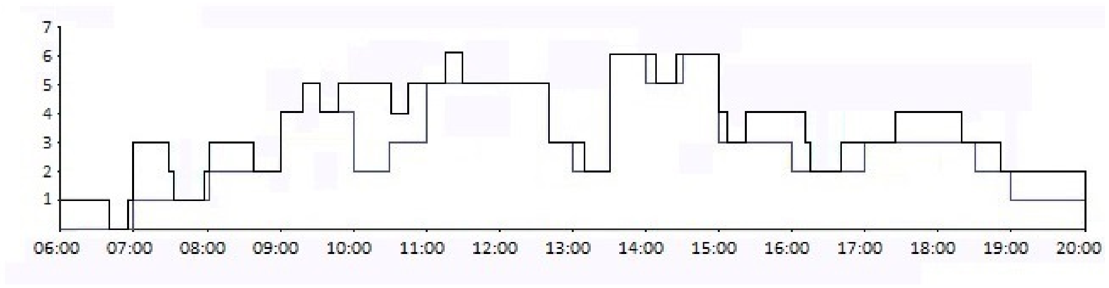


Figure 4.3: Breaks are assigned to each shift, The required employees (Blue Line) and working employees (Black Line) are shown over planning period.

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 $shiftMinusRequirement_t$;
```

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

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.

- 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 constraints 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 exception of this soft constraint is evaluated with the weight W_0 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.

  - C_2 : Work periods are the durations of work between the breaks and also between the first break and start of a shift and the last break and end of a shift. These durations 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, after 50 minutes of the work period, have less than 20 minutes break and the weight value 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 less and more break duration is evaluated with the weight, W_4 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

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

Initialization of Breaks

Each shift-day-duty s have 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 type). These three break types are bl_{sbt}, l_{sbt}, 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 these intervals. These three break type variables are assigned true, if there exists a start of a break in the time slot t.

$$bl_{sbt} = \begin{cases} 1 & \text{if shift-day-duty } s \text{ have break } b \text{ before lunch, starts from time slot } t \\ 0 & \text{otherwise} \end{cases} \tag{4.28}$$

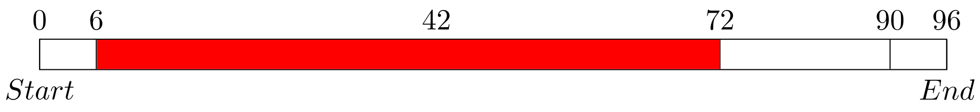
$$l_{st} = \begin{cases} 1 & \text{if shift-day-duty } s \text{ have lunch break, starts from time slot } t \\ 0 & \text{otherwise} \end{cases} \tag{4.29}$$

$$al_{sbt} = \begin{cases} 1 & \text{if shift-day-duty } s \text{ have break } b \text{ after lunch, starts from time slot } t \\ 0 & \text{otherwise} \end{cases} \tag{4.30}$$

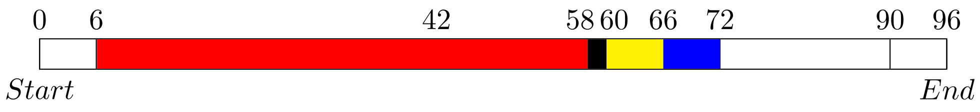
These time slots are restricted with different time windows, based on types of breaks. We are assuming that every staff has three monitor breaks before the lunch break and $m - 4$ monitor breaks after the lunch break depends on $s_i.breakTime$ of shift s_i. We will present them as follows, and explain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

- bl_{sbt} : We assumed each shift-day-duty s have 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 earliest start of the before lunch breaks, based on constraint C_0, at earliest start of the break *earliestStart*.
- These breaks are before the lunch break, therefore the latest start of a before lunch breaks can be the latest end of lunch break of a shift, 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 these breaks and before the latest end of lunch break. The employees can have their lunch break as latest in the end of the lunch break period (last 6 time slots) 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 time slot t is the start time of the break, we need to decrease 2 more time slots from the latest start of the before lunch breaks. The illustrated example of the restricted time window of before lunch break with shift length 8 is shown below (Red: Time Window of Before Lunch Break, Black: Monitor Break, Yellow: Minimum Working Period, Blue: Lunch Break),



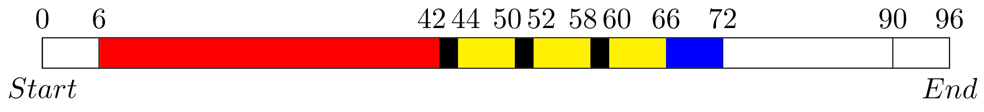
From the restrictions above, the time window of before lunch breaks is calculated below,

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

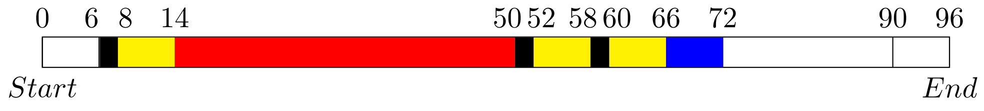
$$ls_0 = lunchLatestEnd - lunchBreakTime - minWP - 2 \tag{4.32}$$

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

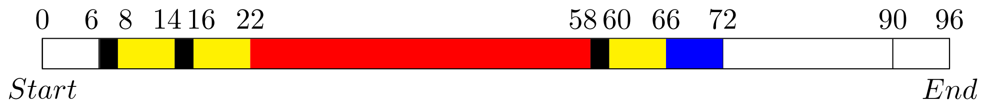
- The earliest start of the first before lunch break is the earliest start of the break *earliestStart*. However, it must end before the other remaining two before lunch breaks. Therefore, we consider the last before lunch break starts at the latest start of the time window above. Before this break the employee need to work *minWP*. The second break 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 to find the latest start of the first before lunch break. The illustrated example of the restricted time window for the first before lunch break 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 second break, that is $minWP + 2$ time slots before the old latest start. The earliest start of the second before lunch break must be after the first break, therefore, we need to add to the first before lunch break, $minWP + 2$ time slots, to the earliest start of the break. As an example of the restricted time window for the second before lunch break 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 third before lunch break must start after $2*(minWP+2)$ from earliest start of the break and latest start is the old latest start, that we calculated above. The illustrated example of the restricted time window for the last before lunch break is given below (Red : Time Window of Before Lunch Break, Black : Monitor Break, Yellow : Minimum Working Period, Blue : Lunch Break),



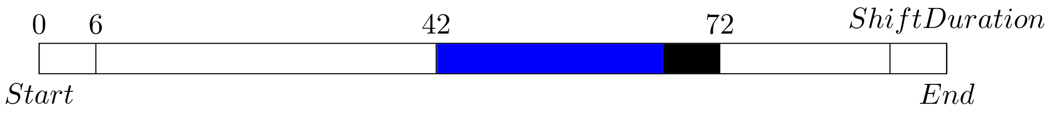
As a summary, we added $i * (minWP + 2)$ to the earliest start of a break and decrease $(2 - i) * (minWP + 2)$ from the latest start where i indicating the number of before lunch break -1. The updated interval of time window TW_0 of before lunch break is given below,

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

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

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

- 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 Lunch Earliest Start and Lunch Latest End - Lunch Break Time. This break has length 6 time slots. The illustration of time window TW_1 of lunch break is shown below, (Blue: Time Window of Lunch Break, Black: Lunch Break),



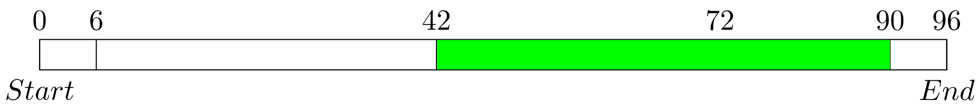
The earliest start and latest start of a lunch break is given below,

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

$$ls_1 = LunchLatestEnd - LunchBreakTime \tag{4.37}$$

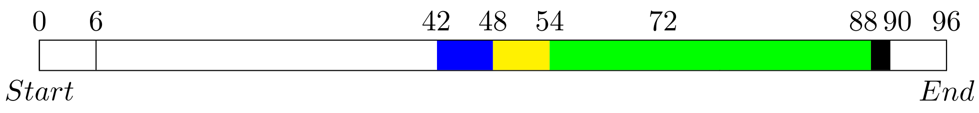
- al_{sbt} : We assumed each shift-day-duty s have $m - 4$ monitor breaks after lunch, based on length of the shift. We need to consider the followings to calculate the time window TW_2 of these types of breaks,

  - The latest start of the after lunch breaks, based on constraint C_0, at latest end of the break *latestEnd*.
  - These breaks are after the lunch break, therefore the earliest start can be the earliest start of lunch break of a shift. 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. Therefore, assuming 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 time slot t is the start time of the break, we need to decrease 2 time slot from the latest end of the break *latestEnd*. The illustrated example of the restricted time window of after lunch break with shift length 8 is shown below (Green : Time Window of After Lunch Break, Black : Monitor Break, Yellow : Minimum

Working Period, Blue : Lunch Break),



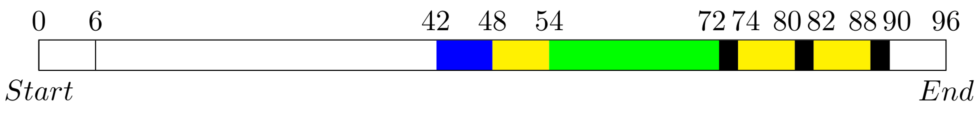
As we explained in before lunch breaks, we can also restrict the time window of each after lunch break TW_2 between each other. We add i * (minWP +2) to earliest start of after lunch break and decrease (m - 5- i) * (minWP +2) from latest start of after lunch break where i indicating the number of after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

$$es_2 = LunchEarliestStart + LunchBreakTime + minWP * (i + 1) + 2i \quad (4.38)$$

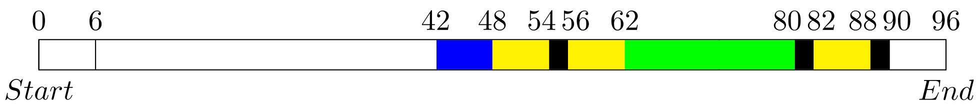
$$ls_2 = LatestEnd - 2 - (m - 5 - i) * (2 + minWP) \quad (4.39)$$

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

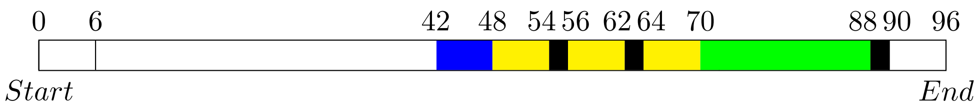
The illustrated example of the restricted time window for the first after lunch break with shift length 8 is given below (Green: Time Window of After Lunch Break, Black: Monitor Break, Yellow: Minimum Working Period, Blue: Lunch Break),



The second after lunch break with shift length 8 is shown below (Green: Time Window of After Lunch Break, Black: Monitor Break, Yellow: Minimum Working Period, Blue: Lunch Break),



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



- st_{sb} : Start time is an integer positive variable, indicating 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,

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

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

  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 each work period have 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,

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

  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},

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

  - For lunch breaks l_{st},

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

  - For after lunch breaks al_{sbt},

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

- st_{sb} : Start time variable indicates the time difference between the break b start 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,

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

- 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,

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

- 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,

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

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, 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 (5. - 11. - 19. Lines) to get each type of breaks 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.

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

- Sum of excesses of employees in each time slot

$$C_5 = \sum_{t=0}^{n-1} ex_t \tag{4.55}$$

- Sum of shortages of employees in each time slot

$$C_6 = \sum_{t=0}^{n-1} sh_t \tag{4.56}$$

4.2.3 Objective function

The objective function is combined two remaining weighted criteria

$$\min \sum_{i=5}^{6} W_i * C_i \tag{4.57}$$

Algorithm 4.2: Algorithm to calculate sum of all employees have break in time slot t

```
Input: $bl_{sbt}$, $l_{st}$, $al_{sbt}$, $s_i.start$, $shiftMinusRequirement_t$ ..
Output: $sh_t, ex_t$
for $i \leftarrow 0$ to $n-1$ do
    for $s \leftarrow 0$ to $sdd-1$ do
        for $t \in TW_0$ do
            for $b \leftarrow 0$ to $2$ do
                if $i = (t+s_i.start+earliestStart+(minWP+2)*b+(daySDD.get(s)*n/7)) \mod n$ then
                    $breaks_i+ = bl_{sbt} + bl_{sb(t-1)} + bl_{sb(t-2)}$ ;
                end
            end
        end
        for $t \in TW_1$ do
            if $i = (t + s_i.start + lunchEarliestStart + (daySDD.get(s) * n/7)) \mod n$ then
                for $j \leftarrow 0$ to $6$ do
                    $breaks_i+ = l_{s(t-j)}$ ;
                end
            end
        end
        for $t \in TW_2$ do
            for $b \leftarrow 0$ to $m-5$ do
                if $i = (t + s_i.start + lunchEarliestStart + 6 + minWP + (minWP + 2) * b + (daySDD.get(s) * n/7)) \mod n$ then
                    $breaks_i+ = al_{sbt} + al_{sb(t-1)} + al_{sb(t-2)}$ ;
                end
            end
        end
    end
    $breaks_i - sh_i + ex_i = shiftMinusRequirement_i$ ;
end
return $sh_t, ex_t$;
```

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 we will give further details separately in two subsections based on our two problems. In each subsection, first of all different benchmark instances for both problems will be introduced and afterwards present the experiments based on different parameters or ILP solver. In the last stage the results will be compared with the best result, that obtained up to date.

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. We also experiment with Gurobi Solver version 5.6.3 and its performance is compared with the Cplex Solver for minimum shift design problem. We have used Cplex Solver with default parameters in the first step, however, in some instances, which take too long times, we have changed some parameters to terminate the branch and bound to decrease run time. This change will be explained in details.

All experiments were performed on a machine, 2.4GHz Intel Core i5 CPU and 8.00 GB of RAM

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 best solutions in the state-of-the-art.

5.1.1 Description of the Instances

We have used the four different sets of problem instances for our experiments. The detail information about the data sets can be obtained and the instances can be downloaded from the link below,

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

These first three benchmark set is randomly generated and the data set 4 is the only one, that have data of real life instance. These data sets are the only example for the shift design problem, that has been first introduced in [MSS04] [Mus01]. The further investigations are used in these instances to compare their results in the literature ([MSS04] [Abs13] [DGGK$^+$07]).

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 similar to the first data set. There are not any over- or under-cover of workers. However, the first 10 instances (1-10) 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. The relation between shift number can be tested with these instances in this data set.

Data Set 3

The third set is generated same as first two datasets, however, shifts are constructed also invalid up to 4 time slots, with deviating from normal starting times and lengths. The same number of shifts (valid and invalid shifts) used as in data set 2. These invalid shifts end up with deviation solutions, due to the existence of under- or over-staffing. Therefore, this data set is significant to compare approaches.

Data Set 4

The last data set consist of three instances, the first one is a real life instance, taken from a call center, instead of randomly generated ones. 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 have 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 are easier than the third data set.

In Table 5.1 and Table 5.2, we present the optimum 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 run time of the existing result will be discussed in the next section for data set 1. 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 the optimal solutions for data set 1.

As shown in the Table 5.2, the instance number 27 in data set 2, we terminated after the run reached some amount of time in Cplex Solver. To solve this instance, we will change the default parameter of Cplex, with restricting the run time or the gap between the optimal solution and the founded best integer solution. Normally the Cplex run until this gap is equal to 0 percent, that prove the optimality. 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.) because the run time exceeds over 2 hours time limit.

The third data sets are generated with invalid shifts. This cause excesses and shortages of employees in each time slot. The complexity of the problem is increased and to reach the optimal solution become harder. However, we have found really good results, compared to the existing results in the literature.

For data set 3, we present our solution in two tables based on integer linear programming solver with 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). We also run this data set once in both ilp solvers and for a 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 5 instances with 2 hours time limit and the run takes longer time than Cplex Solver. In several instances, Gurobi Solver finds the optimal solutions differently from Cplex Solver. That is, Gurobi Solver obtained not only fewer or more under- and over-staffing, but also one fewer or one more number of shifts, although the objective function values are the same.

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 5. instance of dataset 3. Therefore, we added the result of it in the last column to show difference easier.

Instance	Best	Nr. Of	Cplex	Gurobi	Existing	Resource
Instance	Solution	Shifts			Solutions	
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) to reach the best known solution using Cplex and Gurobi Solver and best solution in the state-of-the-art for Data Set 1

Instance	Best Solution Solution	Nr. Of Shifts Shifts	Cplex	Gurobi
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	Best Solution	Nr. Of Shifts	Nr. Of Undercover	Nr. Of Overcover	CPU Time
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	4605	25	2370	1860	14604
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) to reach the best known solution using Cplex Solver for Data Set 3

Instance	Best Solution	Nr. Of Shifts	Nr. Of Undercover	Nr. Of Overcover	CPU Time
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	8910	18	7320	1050	9817
21					> 7200
22	12600	14	4050	8130	16505
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) to reach the best known solution using Gurobi Solver for Data Set 3

Instance	Best Solution	Nr. Of Shifts	Nr. Of Undercover	Nr. Of Overcover	CPU Time
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) to reach the best known solution using Cplex Solver for Data Set 4 and 5. instance of Data Set 3

Instance	Best Solution	Nr. Of Shifts	Nr. Of Undercover	Nr. Of Overcover	CPU Time
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) to reach the best known solution using Gurobi Solver for Data Set 4 and 5. instance of Data Set 3

The real life instance (4-1) run only one second and found 12 feasible shifts. 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 function have the same value with different under- and over-staffing and take a lot more time. The third one have 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 obtain 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 Instance 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 have the optimal values, however Gurobi Solver could not obtain optimal solutions for more instances due to two hours run time limit. Therefore, we will prefer to use Cplex Solver in the further runs.

5.1.3 Computational Results with some Limitation in Cplex Parameters

For our experimentations, we have used 3 different parameters in Cplex Solver. These parameters speed up the process, meanwhile change the optimality tolerance. One of these parameters is a time limit to terminate the ILP-Solver and the remaining two other parameters are related to gap value between the best founded 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 set a certain percentage of gap to the optimal solution. This parameter can be between 0 and 1.

- EpAGap : Absolute optimality tolerance 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, thus small numbers are less useful in optimality tolerance.

These parameters are illustrated 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 result of the instances, that are over the time limit in data set 2 and data set 3, 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 30. We have tried different values for EpGap parameter for the instance in data set 2 and instances in data set 3. Because, data set 3 has also thousands of shortages and excesses of workers and it increases the objective functions. The value of EpGap is related with the objective function.

For instance, the optimum value of instance 27 in data set 2 is 300, according to [Abs13]. If we use the EpGap value is equal to % 2, the gap is set around 6 for optimum value around 300. As we know that in data set 2, the objective function 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 of 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 more gap value ends up with more objective function value and the less gap does not change the result and increase the run time. Meanwhile, we have obtained 315 as a result of the instance 27 in data set 2, however, this result is not the optimum solution considering to the article [Abs13].

We will use the best results, that we achieve in this section, to compare our results with the existing state of the art's results in the literature in following part.

5.1.4 Comparison with Existing Results

In this section, we will compare our results, that we achieved the best results for each instance with the best existing result for each instance in the state of the art, these previous results are investigated by Di Gaspero et al. in [DGGK$^+$07], Musliu et al. in [MSS04] and in master thesis of Abseher in [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 environment of these machines to compare with our solution. In [MSS04] [DGGK$^+$07] [Abs13] , the following approaches were proposed:

Instance	Paramater	Best Solution	Time	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) and Results using different parameters in Cplex Solver for Data Set 4 and 5. instance of Data Set 3

- LS : A local search solvers with a set of move types to explore the neighbourhood, is proposed by Musliu et al. [MSS04]. In order to avoid cycles in the move selection process, tabu search mechanism is used.

- 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. Shifts are edges and workforces are the edge flows. 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 performance could not be improved obtained by solving the shift design problem using the heuristic-based approaches in [DGGK$^+$07]. Therefore, we compare these works 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 overstaffing. Therefore, the optimum results are achieved easily for existing results in the state-of-the-art. 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.

Cplex Solver obtain the optimal solution with a less run time compared to the existing approaches in the state-of-the-art, Only in instance number 25 in data set 1, Abseher found the solution in shorter 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. The instances 13 - 19 - 21, we have used Cplex parameters, therefore we are not able to know that the results for these instances are optimal, 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	2385	2,445.00	9,916.35	2,386.80
2	7590	7,672.59	9,582.00	7,691.40
3	9540	9,582.14	12,367.50	9,597.00
4	6540	6,634.40	8,956.50	6,681.60
5	9720	10,053.75	10,311.60	9,996.00
6	2070	2,082.17	4,712.25	2,076.75
7	6075	6,075.00	12,251.70	6,087.00
8	8580	9,023.46	10,512.60	8,860.50
9	6000	6,039.18	11,640.60	6,036.90
10	2940	2,968.95	4,067.10	3,002.40
11	5190	5,511.43	7,888.20	5,490.90
12	4110	4,231.96	11,410.05	4,171.20
13	4605	4,669.50	10,427.55	4,662.00
14	9600	9,616.55	10,130.40	9,660.60
15	11250	11,448.90	13,563.60	11,445.00
16	10620	10,785.00	11,180.40	10,734.00
17	4680	4,746.56	11,735.40	4,729.05
18	6540	6,769.41	9,516.60	6,692.40
19	4890	5,183.16	10,825.20	5,157.45
20	8910	9,153.90	12,481.80	9,174.90
21	5925	6,072.86	14,102.55	6,053.55
22	12600	12,932.31	16,418.70	12,870.30
23	8280	8,384.24	9,788.40	8,390.40
24	10260	10,545.00	11,413.20	10,417.80
25	13020	13,204.80	14,038.80	13,252.20
26	12780	13,152.73	17,326.50	13,117.80
27	10020	10,084.94	10,866.60	10,081.20
28	10440	10,641.21	11,543.40	10,603.80
29	6510	6,799.41	12,075.30	6,690.00
30	13320	13,770.68	14,808.60	13,723.80

Table 5.8: Comparison of Solution Costs for Data Set 3

5.2 Break Scheduling

For the computational result of the break scheduling 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 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 detail information about the data sets can be obtained and the instances can be downloaded from the link below, http://www.dbai.tuwien.ac.at/proj/SoftNet/Supervision/Benchmarks/

These instances are generated from the benchmark of the shift design problem. Afterwards, break time is computed for each shift, and breaks are added between 10 to 20 minutes to the instance of the shift design problem, with satisfying 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 to these calculated number of working employees included breaks.

There are also real life instances, that have used in existing result of break scheduling problem. There are not any information about these real life instances on the internet.

We have tried to run these instances also in our integer linear programming approach, however, our formulation is not convenient for these instances. Because, 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. Shift with a length of 12 hours consists of 36 number of time slots breaks with slot length 5. 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 30 minutes.

Regarding all these constraints, our formulation is not sufficient with the long shifts. We will leave this part as a future work and obtain and compare our results for just the random example for break scheduling, that are used in previous results in state-of-the-art [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. The random instances, that have been used in existing result in the literature, are run to compare in the next stage. In this section, we will show our result for the several instances in 2 randomly generated data sets in Table **??**.

For break scheduling problem, we have restricted the problem and used only the sum of excesses and shortages of workers in each time slot in the objective function, the variables are initialized as supposing the other soft constraints. We present in Table **??**, the violations of these two remaining soft constraints, optimal solution and run time of the random generated instances.

We have restricted our runs with two hours time limit and some of these instances found the optimal solutions before the time limit. However, these optimal solutions are found with the restricted formulation of break scheduling problem. It can be possible that optimal schedule have more or fewer breaks before lunch or more or less break time instead of each of them have 10 minutes length ..etc. For this reason, we can not guarantee optimality for break scheduling examples. The other ones, that run time is more than one hour we found feasible solutions, we can not be sure that these are optimal or not.

Instance	Shifts(sdd) Solution	Best	Time Undercover	Nr. Of Overcover	Nr. Of
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: The result of break scheduling problem with randomly generated instance

Instance	Shifts(sdd)	Cplex	MAPBS [WM14]		
			Best	Avg	σ
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 with literature of break scheduling problem with randomly generated instance

5.2.3 Comparison with Existing Results

In this section, we will compare our results with the best existing result for each instance 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 set for break scheduling problem.

Widl et al. proposed MAPBS approach in [WM14] [Wid10] [WM10] based on memetic algorithm and achieve the best known results for break scheduling problem, in each random generated instance. The experiments, that we will compare with our results, were performed on a machine, 2.33GHz of a QuadCore Intel Xeon 5345 and with a main memory 48 GB. 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 the same seconds.

The comparison of the results is shown in Table 5.10

We obtain optimal solution for most of the instances in randomly generated datasets, for the restricted break scheduling problem formulation. As we stated before, we defined

3 breaks before the lunch and the length of each break 10 minutes and we initialize our instances satisfying the first 5 soft constaints. These restrictions can end up with failing to find optimal solution. For instance, Cplex found the solution of the random1-28 instance, as optimal, however Widl et al. reached a better solution for this example, although our result is smaller than the average of the runs of memetic algorithm.

The instance random1-28 is the only random generated instance, that we could not obtain the best results, but except this random instance, we improved the best solutions for break scheduling problem.

In several instances, we need to terminate our runs after exceeding the time limit. Although, we have found feasible solutions and these results are also generally improved the previous best solutions for break scheduling data set.

As a summary, integer programming formulation for the restricted break scheduling problem with Cplex Solver obtain optimum or the best known solutions for randomly generated data sets . However, we formulated this problem with supposing each shift have 3 breaks before the lunch and this assumption ends up with too many breaks after the lunch break for the 12 hours or longer shifts. These assumptions must be reconsidered for the future works.

CHAPTER 6

Conclusion and FutureWork

6.1 Summary

In this thesis, we developed integer linear programming formulation for solving shift design and break scheduling problems, that belong to class of NP-hard problems. These problems are variants of shift scheduling problem and are introduced recently in the literature. 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.

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. There are different proposed integer programming formulation for shift scheduling problem. Shift design problem is variety of shift scheduling problem. Shift design problem has several criteria, that needs to be considered.The shifts are generated over multiple days, usually a week and has a cyclic structure. In order to minimize the number of shifts in shift design problem, we need to consider reusing shifts on all days of the week. In addition to minimizing the number of shifts, the objective function consists of sum of the shortages and excesses of workers in each time slot.

We solve our integer linear programming for shift design problem with Cplex and Gurobi Solvers, to compare both solvers each other. We reach optimal solution with less time using Cplex Solver. Therefore, we continue our further investigations with Cplex. We have used 4 different randomly generated instance set, including one real life instance. We obtained the optimal solution for most of the instances, however, we needed to terminate a few of instances, due to the exceeding of time limit (2 hours). To speed up these instances, that timed out, we have used three different parameters of Cplex Solver. We obtained better objective function values compared to the best known results in state-of-the-art. However, these changes with parameters does not guarantee the optimality of these instances. All generated instances except one (27. instance in dataset 2), we have found the best known result in the literature.

The second problem, that we investigated integer linear programming formualtion is break scheduling. Due to the constraints based on work period between breaks, we proposed also explicit model. However, to enumerate all possible breaks ends up with large amount of variables and constraints. For this reason, we formulated break scheduling problem with considering some restrictions. These new hard constraints are, assuming each breaks have 10 minutes length and there exist three breaks before the lunch. In addition to these constraints, we initialize variables, that satisfies 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 set randomly generated instances. The real life instances consist of assigned shifts with different shift lengths and for the instances with shift lengths more than 12 hours cause too many after lunch breaks with supposing 3 breaks before lunch. Therefore, we could not obtain solutions for real life instances, due to negative time window, considering the minimum working period between breaks. However, we have obtained the best known solutions for randomly generated instances, with the same time limit as previous results in the state-of-the-art. Only in one instance, Widl et al. obtained better best result, but still the average of their 10 runs are more than our objective function value.

6.2 Future Work

We obtained the best known results in the state-of-the-art for both 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 each shift have 3 breaks before the lunch and this assumption ends up with too many breaks after the lunch break for the 12 hours or longer shifts. As a future work, we need to reconsider these assumptions and could be beneficial to remove the restrictions for break scheduling problem.

- We proposed integer linear programming formulation for both problems separately as two different problems. To solve break scheduling problem, we need to have assigned shifts as an input variable. To schedule these shifts, we need to consider each break. For this reason, as a future work, to investigate a formulation to solve both problems simultanuously can be very helpful to find more feasible schedules in the subject of assigning shifts and breaks for the employees.

- As mentioned by Di Gaspero et al. in [DGGM$^+$13], we can extend the problem with adding the new considerations like employee qualification and task assignment simultaneously. These new improvements would be useful in a scheduling system that can help professional planners and can be considered for future work.

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 EddieâĂŹs 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.

[WM10] M. Widl and N. Musliu. An improved memetic algorithm for break scheduling. *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.