FAKULTÄT FÜR !NFORMATIK
Faculty of Informatics

MaxSAT Modeling and Metaheuristic Methods for the Employee Scheduling Problem

DIPLOMARBEIT

zur Erlangung des akademischen Grades

Diplom-Ingenieur

im Rahmen des Studiums

Software Engineering & Internet Computing

eingereicht von

Felix Winter, BSc.
Matrikelnummer 0825516

an der Fakultät für Informatik
der Technischen Universität Wien
Betreuung: Priv.-Doz. Dr. Nysret Musliu

Wien, 17. August 2016

Felix Winter	Nysret Musliu

FAKULTÄT FÜR !NFORMATIK
Faculty of Informatics

MaxSAT Modeling and Metaheuristic Methods for the Employee Scheduling Problem

DIPLOMA THESIS

submitted in partial fulfillment of the requirements for the degree of

Diplom-Ingenieur

in

Software Engineering & Internet Computing

by

Felix Winter, BSc.
Registration Number 0825516

to the Faculty of Informatics
at the TU Wien
Advisor: Priv.-Doz. Dr. Nysret Musliu

Vienna, 17th August, 2016

Felix Winter	Nysret Musliu

Erklärung zur Verfassung der Arbeit

Felix Winter, BSc.
Flachgasse 48/16 1150 Wien

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

Wien, 17. August 2016

Felix Winter

Acknowledgements

I would like to express my sincere gratitude to my advisor Priv.-Doz. Dr. Nysret Musliu. Writing this thesis would not have been possible without his ongoing support, encouragement and his deep knowledge in the field.

I want to mention that this work was supported by the Austrian Science Fund (FWF): P24814-N23, and I would like to thank Florian Mischek and Emir Demirović who were my project colleagues during the time of writing. Their comments and feedback helped me to improve this thesis in many places.

Deepest gratitude goes to my parents and my brother, who have supported me throughout my life and aided me through many hard times. Without their help, finishing this work would not have been possible.

Last but not least I want to thank my girlfriend Eni for her patience and emotional support during the time of writing.

Kurzfassung

Die Erstellung von Schichtplänen ist in vielen verschiedenen Bereichen wie z.B. im Flugverkehr, im Gesundheitswesen, im Transportwesen und prinzipiell jedem Unternehmen das eine hohe Mitarbeiterzahl aufweist, erforderlich. Dabei bietet ein effizienter Schichtplan nicht nur die Möglichkeit Kosten zu sparen, sondern ist oftmals auch notwendig um eine gesunde Arbeitsumgebung sowie eine ausreichende Zufriedenstellung der Mitarbeiter zu gewährleisten. Obwohl verschiedene heuristische und exakte Verfahren für die automatisierte Erstellung effizienter Schichtpläne bereits seit mehreren Jahrzehnten eingesetzt werden, gibt es weiterhin eine große Nachfrage nach neuen Lösungsansätzen, um bestehende sowie neuaufkommende Problemstellungen aus diesem Gebiet optimal zu lösen.

Gegenstand dieser Diplomarbeit ist die Entwicklung zweier neuer Lösungsmethoden für bekannte Schichtplanprobleme aus der einschlägigen Literatur. Das erste vorgeschlagene Verfahren modelliert bekannte Schichtplanprobleme als MaxSAT-Variante des Erfüllbarkeitsproblems der Aussagenlogik. Verschiedene bestehende Methoden zur Lösung von MaxSAT-Problemen werden in der Folge verwendet, um mithilfe dieses Modells Schichtplanprobleme zu lösen. Als zweiter Lösungsansatz wird ein neuer Hybridalgorithmus beschrieben, der exakte Methoden mit heuristischen Verfahren kombiniert. Dabei kommen sowohl Metaheuristiken als auch Techniken aus der Constraintprogrammierung zum Einsatz.

Beide Lösungsverfahren werden mittels einer Reihe von Experimenten evaluiert. Dabei werden Testresultate zu bekannten Problemstellungen unter Verwendung beider Methoden erzeugt und mit den Ergebnissen bestehender Lösungsansätze aus der Literatur verglichen. Das auf dem MaxSAT-Modell basierende Verfahren kann dabei eine große Zahl an Schichtplanproblemen erfolgreich lösen und die Optimalwerte bekannter Lösungen verifizieren. Mithilfe des in dieser Arbeit vorgeschlagenen Hybridalgorithmus werden viele Ergebnisse, welche durch Methoden aus der Literatur erzielt wurden, verbessert. Weiters werden fünf neue obere Kostschranken für bekannte Schichtplanprobleme ermittelt.

Abstract

Employee scheduling is a well known problem that appears in a wide range of different areas including health care, airlines, transportation services, and basically any other organization that has to deal with workforces. The creation of efficient schedules often becomes essential not only to save avoidable costs, but also to provide a healthy work environment and to satisfy the preferences of the employees. Although different approaches based on heuristics as well as exact solution techniques have been proposed in the past, there is still a great demand for innovative strategies which are able to find good solutions for the many existing variants of employee scheduling problems.

This thesis proposes two novel solution approaches for well known instances of the employee scheduling problem and evaluates their performance. The first approach provides for the first time a weighted partial boolean maximum satisfiability model for employee scheduling. An analysis of different encoding variants is performed and results achieved by leading maximum satisfiability solvers are evaluated.

The second solution approach which is proposed in this thesis introduces a novel hybrid algorithm that combines metaheuristic techniques with exact solution strategies that are based on constraint programming. Using an implementation of the proposed algorithm, a large number of experiments is then conducted on a number of well known problem instances from the literature. The obtained results are compared with the best known solutions which were acquired by state of the art methods.

An empirical evaluation of the proposed methods shows their competitiveness with state of the art solutions. The maximum satisfiability model is used successfully to verify optimal bounds and to produce feasible solutions for most of the considered problem instances. Using the hybrid algorithm that is proposed in the thesis, results produced by state of the art techniques are improved for the majority of the considered problem instances. Additionally, five new unknown upper bounds are provided for the considered problem instances.

Contents

CHAPTER 1

Introduction

Employee shift scheduling problems appear whenever there is the need to efficiently construct a shift roster over a certain time period. This includes a wide range of different areas like health care, air lines, transportation services, armed forces, call centers, emergency services, and basically any other institution that has to deal with workforces. Unfortunately, the process of creating an efficient roster can be very challenging and time consuming. In fact, it has been shown that it belongs to the class of NP-hard problems. The construction of good schedules can easily become a very tedious, sometimes even impossible task for a human and the development of efficient automated solution methods therefore is a topic of active research.

A variety of staff scheduling problems have been described in the past. Surveys about the many different solution methods that have been proposed in the literature can be found in [EJKS04] and [dBBB+13]. The majority of the approaches suggested in the literature rely on mathematical programming and metaheuristic techniques and although such methods have shown to provide good solutions, optimal solutions are still unknown for many problem instances. Therefore, the application of new solution methods has still to be considered.

One example for a method that could be investigated relies on modeling the problem as a maximum satisfiability problem (maxSAT). The intuitive way of working with propositional formulas, as well as growing developments in the SAT community motivate the investigation of such an approach. Furthermore, modeling a problem with a maximum propositional satisfiability formulation has shown to perform well on a variety of different applications in the past, including the scheduling of B2B meetings [BGSV15] and High School Timetabling [DM14]. Another interesting research topic is the development of new hybrid techniques that combine heuristics with exact solution methods for employee scheduling.

The goal of this thesis is to investigate such approaches for the purpose of solving a well known employee problem that is described in [CQ14]. According to the authors those instances were designed to describe realistic and challenging staff scheduling problems. The included instances provide a broad variety of problem scenarios with scheduling periods ranging from one week up to one year and up to 180 employees and 32 shift types that need to be scheduled.

Recent publications provided an integer programming (IP) model as well as a metaheuristic approach to solve those instances. The best known solutions using these techniques as well as a description of the IP formulation can be found in [CQ14]. A detailed description of the applied algorithms, namely a branch and price method and a metaheuristic based on ejection chains can be found in [BC14] and [BCP+08]. With the use of the branch and price algorithm and ejection chains, optimal solutions could be found for most of the smaller instances and lower/upper bounds could be determined for many of the remaining instances. However, optimal solutions are still unknown for a large number of instances.

1.1 Aims of this Thesis

The main objectives of this thesis are:

- Investigation of a new solution approach for employee scheduling that uses a partial weighted maxSAT model.
- Critical evaluation of different modeling strategies and leading maxSAT solvers from the literature.
- Development and implementation of a novel hybrid approach that combines metaheuristic as well as exact solution strategies.
- Evaluation of the proposed hybrid solver and detailed comparison with state of the art solutions.

1.2 Main Results

The main contributions of this thesis are:

- The first maxSAT formulation for the variant of the employee scheduling problem from [CQ14] is provided.
- Experiments with different maxSAT encodings are conducted and results produced by two leading maxSAT solvers from the literature are compared with results produced by state of the art solutions.
- Challenging instances which can be used by the maxSAT community to test and improve results of maxSAT solvers are provided.

- A novel hybrid approach that uses the min-conflicts heuristic and an iterated local search with a constraint programming based perturbation is provided.
- An empirical evaluation of the developed hybrid approach is performed and the produced results are compared to results that were produced by state of the art solutions.
- Results produced by state of the art methods are improved for the majority of the considered problem instances. Additionally, five new upper bounds for well known problem instances from the literature are provided.

1.3 Organization

Chapter 2 introduces the employee scheduling problem and provides an overview of related work from the literature. Additionally, some background on the existing solution strategies that are applied in this thesis will be given. Chapter 3 describes the details of the proposed maxSAT formulation and provides additional information about the used encodings. The details about the novel hybrid search approach are then explained in chapter 4. An experimental evaluation about using leading maxSAT solvers together with the proposed formulation to solve employee scheduling problems is performed in 5. Results of using the proposed hybrid algorithm to solve the problem instances are then reported and compared with state of the art results in chapter 5.3. Finally, concluding remarks are given in chapter 6.

CHAPTER 2

Problem Statement and Related Work

The following chapter will introduce the considered employee scheduling problem. After a detailed problem definition, related work that has been published in the literate will be presented and summarized.

2.1 Problem Definition

This thesis deals with a variant of the employee scheduling problem that is described in [CQ14]. This specific problem formulation has been chosen as it provides a number of instances that include challenging and realistic scheduling problems, while still being intuitive and straightforward to use. Although the problem that is investigated in this thesis is a specific employee scheduling problem, the considered problem will be referred to as employee scheduling problem from now on.

The overall goal of the problem is to find an optimal roster for a number of given employees and shift types, where every employee may either work in a single shift or have a day off on each day of a given scheduling period. For this problem the scheduling period is stated as a number of weeks and therefore the number of days is always a multitude of seven. Another property concerning the scheduling horizon ensures that the first day of the roster is always a Monday, while the last day is always a Sunday. The employees and shift types which are considered in this problem, are specified by a list of unique names which are connected with a number of constraints that restrict all possible shift assignments. Some employees might for example be only allowed to work in certain shifts and patterns of consecutive working shifts might be prohibited or requested. Each problem instance specifies hard- and soft-constraints to set up a corresponding rule set. Hard constraints on the one hand are always strict and have to be fulfilled in order to

generate a feasible solution. Soft constraints on the other hand may be violated, but will in case of a violation, lead to an integer valued penalty. For example one of the problem's hard constraints specifies the minimum and maximum amount of time that an employee can work during the scheduling horizon. Personal shift requests that employees can state are formulated as soft constraints in the problem instances.

In the following a detailed definition of the problem's input parameters, constraints and evaluation function will be given. This problem definition is based on [CQ14].

2.1.1 Problem Parameters

I	set of employees.
h	number of days in the planning horizon
D	set of days in the planning horizon = $\{1...h\}$
W	set of weekends in the planning horizon (day indices of all Saturdays)
T	set of shift types
R_t	set of shift types that cannot be assigned immediately after shift type t
N_i	set of days that employee i cannot be assigned a shift on
l_t	length of shift type t in minutes
m_{it}^{max}	maximum number of shifts of type t that can be assigned to employee i
b_i^{min}	minimum number of minutes that employee i must work
b_i^{max}	maximum number of minutes that employee i may work
c_i^{min}	minimum number of consecutive shifts that employee i must work
c_i^{max}	maximum number of consecutive shifts that employee i may work
o_i^{min}	minimum number of consecutive days off that may be assigned to employee i
a_i^{max}	maximum number of weekends that employee i may work on
q_{idt}	penalty which is given, if shift type t is not assigned to employee i on day d
p_{idt}	penalty which is given, if shift type t is assigned to employee i on day d
u_{dt}	preferred total number of employees that should have shift type t assigned on day d
v_{dt}^{min}	penalty which is given, if below the preferred cover for shift type t on day d
v_{dt}^{max}	penalty which is given, if exceeding the preferred cover for shift type t on day d

2.1.2 Hard Constraints

Several hard constraints are given with an instance of the employee scheduling problem. The different types of hard constraints that may appear are listed here:

Employees can not be assigned more than one shift on a single day. This constraint ensures that each employee cannot work in more than one shift at the same time.

Disallowed shift sequences. Any shift t may define a set of shift types R_t of forbidden successors. Since each shift type has fixed starting and ending times, this constraint can be used for example to forbid shift sequences that would not guarantee enough resting time between two shifts (e.g. a morning shift should not immediately follow after a night shift).

The maximum number of shifts for each type that can be assigned to an employee. The contracts of an employee might set work limits regarding certain shift types. For all employees i the maximum number of shifts of type t that they may work in during the whole scheduling horizon is defined by the parameter m_{it}^{max}.

Minimum and maximum working time. Each shift type assigns a certain amount of working time in minutes to its associated employees. The total working time in minutes is restricted for each employee and must lie between a minimum and maximum bound. Those limits are given to the problem with the parameters b_i^{min} and b_i^{max} for each $i \in I$.

Maximum consecutive shifts. All employees are only allowed to work for a maximum number of consecutive days before they must have a day off. This maximum limit is given to the problem as c_i^{max} for each $i \in I$. Note that this constraint always assumes a day off before and after the scheduling horizon.

Minimum consecutive shifts. The problem requires that each employee always works on at least a minimum number of consecutive days. In other words there are lower bounds regarding the allowed number of consecutive shifts, which are given as parameter c_i^{min} for all $i \in I$, that the employees must work in before they are allowed to have a day off. Note that this constraint always assumes an infinite number of consecutive working days before and after the scheduling horizon.

Minimum consecutive days off. Whenever an employee has a day off after a day of work, this employee must have a minimum number of consecutive work free days assigned on after the work day. In other words, there is a minimum number of consecutive days off for each employee i, which is defined with the parameter o_i^{min}. Note that this constraint always assumes an infinite number of days off before and after the scheduling horizon.

Maximum number of weekends. Whenever an employee has to work in a shift on a Saturday or a Sunday in the schedule, the corresponding weekend is considered as a working weekend for this employee. The problem restricts the number of such working weekends for each employee i with the parameter a_i^{max}.

Days off. All employees may have certain days on which it is strictly required that they have a day off. Those are given to the problem as a set of day indices N_i for each employee i.

2.1.3 Soft Constraints

Additional to the previously described hard constraints, the considered employee scheduling problem also defines a number of soft constraints. Those constraints do not have to be necessarily fulfilled, but will lead to an integer valued penalty that influences the solution cost if they are violated.

Requested shift types. All employees may specify some days where they request to work in a certain shift type. Since this is not a hard constraint, any violation will be penalized. The corresponding penalties are given to the problem as parameters $q_{i,d,t}$, where $i \in I$, $d \in D$ and $t \in T$.

Unpreferred shift types. Similar as with the requested shift types constraint, the problem may contain requests where an employee requires not to work in particular shift types on certain days. The corresponding penalties are given to the problem as parameters $p_{i,d,t}$, where $i \in I$, $d \in D$ and $t \in T$.

Cover requirements. The preferred numbers of employees that work in a shift t on day d is given to the problem in form of parameters u_{dt} for all $d \in D$ and $t \in T$. Furthermore, for each of those values, two penalty parameters v_{dt}^{min} and v_{dt}^{max} are used to penalize a possible under- or over-coverage of the preferred value. Note that penalties will increase linearly dependent on the actual difference to the preferred value. For example, if five employees work in a night shift on day one in the schedule, $u_{1N} = 7$ and $v_{1N}^{min} = 100$, a penalty of 200 would be given because of two missing night shift workers.

2.2 Related Work

Many different staff scheduling problems have been described in the literature and a variety of solution methods have been proposed to approach them. Corresponding surveys regarding employee scheduling can be found in [EJKS04] and [dBBB+13]. In [dBBB+13], approaches on staff scheduling are categorized in seven groups of methods: mathematical programming, constructive heuristics, improvement heuristics, simulation, constraint programming, queuing, and others. However, the survey concludes that most publications in the area rely on mathematical programming, metaheuristic as well as the hybridization of both techniques. While exact approaches benefit from the fact that they can provide optimal solutions as well as bounds to any scheduling problem, metaheuristics are often able to produce good solutions fast and will become useful especially in situations where mathematical programming methods can not solve a problem at all within a given time limit. Although no approach that uses a maxSAT model on staff scheduling has been proposed in the literature yet, the successful use of SAT solvers has been described in the literature for similar problems like high school timetabling [DM14] and the scheduling of business meetings [BGSV15].

A number of real life related employee scheduling problem instances that have been proposed in the literature are presented at [Cur14]. Almost all of those problems have already been solved to optimality using heuristics as well as mathematical programming techniques (e.g. [BLQ10], [PGFP09]) and are therefore not further discussed in this thesis.

The problem instances considered in this thesis are described in [CQ14] and were according to the authors designed to provide a number of challenging and realistic problem scenarios. The same authors propose an IP formulation in addition with a branch and price algorithm from [BC14] to find solutions to those instances. Although the IP model could provide optimal solutions and lower bounds for parts of the smaller instances using the branch and price algorithm and Gurobi [GO15], there are still many instances that have not been solved to optimality. Furthermore, exact methods could not find any feasible solution within a one hour time limit for the largest instances. Therefore, a metaheuristic approach from [BC14] which is based on variable neighborhood search and ejection chains has been applied on the problem instances and could provide solutions also for the larger instances.

In the following subsections the two existing approaches for this employee scheduling problem will be presented in further detail. Section 2.2.1 introduces the IP formulation from [CQ14] and section 2.2.2 will shortly describe the methods used by the ejection chain based metaheuristic which is proposed in [BC14].

2.2.1 An IP Formulation for the Employee Scheduling Problem

This section will shortly present an integer programming model for the employee scheduling proposed in [CQ14]. The model uses the following decision variables:

x_{idt} is set to 1, whenever an employee i has shift type t assigned on day d, otherwise it is set to 0.
k_{iw} is set to 1, whenever an employee i works on weekend w, otherwise it is set to 0.
y_{dt} denotes the total value below the preferred cover for shift type t on day d.
z_{dt} denotes the total value above the preferred cover for shift type t on day d.

The constraints are modeled as follows:

Employees can not be assigned more than one shift on a single day.

$$\sum_{t \in T} x_{idt} \leq 1, \forall i \in I, d \in D \tag{2.1}$$

Disallowed shift sequences.

$$x_{idt} + x_{i(d+t)u} \leq 1, \forall i \in I, d \in 1, ..., h-1, t \in T, u \in R_t \tag{2.2}$$

The maximum number of shifts for each type that can be assigned to an employee.

$$\sum_{d \in D} x_{idt} \leq m_{it}^{max}, \forall i \in I, t \in T \tag{2.3}$$

Minimum and maximum working time.

$$b_i^{min} \leq \sum_{d \in D} \sum_{t \in T} l_t x_{idt} \leq b_i^{max}, \forall i \in I \tag{2.4}$$

Maximum consecutive shifts.

$$\sum_{j=d}^{d+c_i^{max}} \sum_{t \in T} x_{ijt} \leq c_i^{max}, \forall i \in I, d \in \{1, ..., h - c_i^{max}\} \tag{2.5}$$

Minimum consecutive shifts.

$$\begin{gathered} \sum_{t \in T} x_{idt} + \left(s - \sum_{j=d+1}^{d+s} \sum_{t \in T} x_{ijt} \right) + \sum_{t \in T} x_{i(d+s+1)t} > 0, \\ \forall i \in I, s \in \{1, ..., c_i^{min} - 1\}, d \in \{1, ..., h - (s+1)\} \end{gathered} \tag{2.6}$$

Minimum consecutive days off.

$$\begin{gathered} \left(1 - \sum_{t \in T} x_{idt} \right) + \sum_{j=d+1}^{d+s} \sum_{t \in T} x_{ijt} + \left(1 - \sum_{t \in T} x_{i(d+s+1)t} \right) > 0, \\ \forall i \in I, s \in \{1, ..., o_i^{min} - 1\}, d \in \{1, ..., h - (s+1)\} \end{gathered} \tag{2.7}$$

Maximum number of weekends.

$$\begin{gathered} k_{iw} \leq \sum_{t \in T} x_{i(7w-1)t} + \sum_{t \in T} x_{i(7w)t} \leq 2k_{iw}, \forall i \in I, w \in W \\ \sum_{w \in W} k_{iw} \leq a_i^{max}, \forall i \in I \end{gathered} \tag{2.8}$$

Days off.

$$x_{idt} = 0, \forall i \in, d \in N_i, t \in Tn \tag{2.9}$$

Cover requirements.

$$\sum_{i \in I} x_{idt} - z_{dt} + y_{dt} = u_{dt}, \forall d \in D, t \in T \tag{2.10}$$

The objective function is defined as follows:

$$Minimize \sum_{i \in I} \sum_{d \in D} \sum_{t \in T} q_{idt}(1 - x_{idt}) + \sum_{i \in I} \sum_{d \in D} \sum_{t \in T} p_{idt} x_{idt} + \sum_{d \in D} \sum_{t \in T} y_{dt} v_{dt}^{min} + \sum_{d \in D} \sum_{t \in T} z_{dt} v_{dt}^{max} \tag{2.11}$$

Results using this IP model together with a branch and price algorithm and Gurobi will compared with the methods proposed in this thesis in chapter 5.

2.2.2 A Heuristic Approach based on Ejection Chains for Employee Scheduling

An heuristic approach that is based on ejection chains has been proposed for the employee scheduling problem in [BC14]. This algorithm applies a variable depth search (VDS) which has been introduced in [BCQB13], and although the core of the method uses ejection chains, there are many other techniques included in the algorithm. An iterative construction mechanism which is based on dynamic programming is used at the start of the algorithm to generate initial solutions at the start of search. Three different hill climbing methods from [BCQB10] which include assignment and de-assignment of shifts, as well as vertical and horizontal swapping of shifts are then incorporated into the algorithm. The same neighborhoods are utilized in the hybrid solver which is proposed in this thesis and will be further described in section 4.3. If the search gets stuck in a local optimum, the VDS algorithm tries to chain together multiple moves to a single larger move. This extended neighborhood might then be able to escape the local optimum. Additionally, a disruption and repair method from [BCP+08] is used if no progress can be made for a certain amount of moves and the time limit is not exceeded. Finally, the heuristic approach also includes a restart mechanism where the schedules of selected employees will be cleared and refilled by a dynamic programming method.

2.3 Background on applied Solution Techniques

The following sections give background information on some of the methods that proposed in this thesis. The provided explanations are kept short and will only give an idea about the concepts of well known techniques and how they can be used to approach the employee scheduling problem. The interested reader is therefore advised to refer to the literature [RvBW06, HS04] in order to get a deeper understanding about those techniques.

2.3.1 Constraint Programming

Methods using constraint programming [RvBW06] describe a given problem by a set of variables, their domains and constraints over these variables. The general approach to find a solution with constraint programming (CP) lies in intelligently enumerating variable assignments and checking their feasibility. In order to reduce the potentially large number of possible assignments to check, different techniques have been proposed. Typically, constraint propagation reduces the domain of variables by removing values

which cannot be part of the solution. Further, the assignment of variables is performed by tree search that includes different pruning strategies like backtracking, forward checking with dynamic ordering etc. In case of the employee scheduling problem, a variable can be generated for each pair $(i, d), i \in I, d \in D$. The goal is then to assign each of those variables a value from the domain T without violating any hard constraint so that the costs produced by violated soft constraints are minimized.

The approach proposed in chapter 4 uses such a formulation together with forward checking search to solve the employee scheduling problem. Details about the proposed constraint programming based algorithm for employee scheduling can be found in section 4.5.

2.3.2 Local Search

Large NP-hard problems, like the employee scheduling problem, usually cannot be solved within feasible time by exact techniques. For such problems typically heuristic techniques are used that give some solutions in reasonable time, but do not guarantee optimality. In this thesis heuristic techniques that are based on local search [HS04] are proposed. Such methods start from an initially generated solution and then try to improve it by iteratively applying small modifications. Since such a procedure only explores neighborhood solutions in each step, the search usually converges fast to a solution which is locally optimal. Therefore, different meta-heuristic techniques that include mechanisms to escape from local optima like simulated annealing, tabu search, etc. have been proposed.

In chapter 4 a hybrid algorithm is proposed that combines constraint programming based methods together with local search to solve the employee scheduling problem. Section 4.4 describes the implementation of the applied local search techniques in detail.

2.3.3 Using a MaxSAT Formulation to solve the Employee Scheduling Problem

Another way of approaching the employee scheduling problem is to model the problem as a maxSAT problem and to use existing solvers from the literature to find solutions to the corresponding maxSAT formulation.

The Maximum Satisfiability Problem

The SAT problem is a decision problem which asks whether there exist assignments of truth values to variables, such that a propositional logic formula is evaluated as true (that is, the formula is satisfied). A propositional logic formula is built from Boolean variables using logical operators and parentheses. The formula is usually given in conjuctive normal form (CNF), meaning that the formula is a conjunction of clauses, where a clause is a disjunction of literals and a literal is a variable or its negation. For example, the formula $(X_1 \vee X_2) \wedge (\neg X_1 \vee \neg X_3)$ is said to be satisfiable, because the assignment $(X_1, X_2, X_3) = (true, false, false)$ satisfies the formula. However, if the

clause $(\neg X_1 \vee X_2 \vee X_3)$ would be inserted, the same assignment would no longer satisfy the formula.

An extension to SAT that is considered in this chapter is partial weighted maxSAT. For maxSAT, clauses are partitioned into hard- and soft clauses, where each soft clause has a weight assigned to it. The goal is to find an assignment which satisfies the hard clauses and at the same time minimizes the sum of weights of the unsatisfied soft clauses. For more in depth information about SAT, maxSAT and the internals of corresponding solver algorithms the interested reader is directed to [BHvMW09].

In chapter 3 the employee scheduling problem will be modeled as a partial weighted maxSAT problem. The obtained maxSAT formulas which model the problem are called encodings. Chapter 5 then gives details about experiments that have been conducted using two maxSAT solvers from the literature.

CHAPTER 3

Modeling the Employee Scheduling Problem as Partial Weighted MaxSAT

In the following chapter, a partial weighted maxSAT model for the employee scheduling problem will be proposed. The Boolean decision variables will be defined in section 3.1. The notion of cardinality constraints is described in section 3.2. Sections 3.3 and 3.3.1 will then explain how all the problem's hard and soft constraints are modeled as maxSAT clauses.

3.1 Decision Variables

In order to model the assignment of shifts to employees, the variables $S_{i,d,t}, \forall i \in I, d \in D, t \in T$ are defined, where I denotes the set of all employees, D refers to the set of days in the planning horizon, and T is the set of all shift types in the problem. Each variable $S_{i,d,t}$ will be set to true if and only if employee i gets the shift type t assigned on the d-th day in the roster, otherwise it will be set to false. Additionally, helper variables $X_{i,d}, \forall i \in I, d \in D$ which are set to true if employee i has no shift assigned on day d are defined. So $X_{i,d}$ is set to true if and only if employee i is considered to have a day off on this day.

To connect the X variables with the decision variables S, the following equivalences are included in the formulation:

$$X_{i,d} \leftrightarrow \bigwedge_{t \in T} \neg S_{i,d,t} \qquad \forall i \in I, d \in D \tag{3.1}$$

In the following sections it is described how each of the problem's constraints are encoded in the partial weighted maxSAT formulation as clauses. Clauses which are generated for soft constraints will additionally have weights assigned.

Since many of the constraints contain properties of cardinality constraints, the notion of a cardinality constraint is introduced shortly in the next section.

3.2 Cardinality Constraints

In order to be able to formulate all of the constraints for the problem, it is necessary to make use of cardinality constraints. Such constraints define limits on the number of truth assignments on a set of given Boolean variables. There are three different types of cardinality constraints: $atLeast_k(x_i : x_i \in X)$, $exactly_k(x_i : x_i \in X)$, and $atMost_k(x_i : x_i \in X)$ which are defined on sets of variables that should have at least, exactly, or at most k variables having their truth value assigned. For example if a cardinality constraint limits the number of true valued variables of the set x_1, x_2, x_3 to $atMost_2(\{x_1, x_2, x_3\})$, the assignment $(x_1, x_2, x_3) = (1, 1, 0)$ is considered to be feasible, while the assignment $(x_1, x_2, x_3) = (1, 1, 1)$ would be considered as infeasible.

Additionally, hard- and soft cardinality constraints have to be distinguished. While hard cardinality constraints decide whether or not the overall solution will become feasible, soft cardinality constraints will only penalize the objective function if violated. In the case of the employee scheduling problem, soft cardinality constraints have a weight assigned and the total penalty is linearly dependent on the difference to the requested number of truth assignments. For example, if the constraint $atLeast_2(\{x_1, x_2, x_3\})$ has a weight of 40, the assignment $(x_1, x_2, x_3) = (0, 0, 0)$ would lead to a penalty of $40 \cdot 2 = 80$.

Different variants of dealing with cardinality constraints in Boolean satisfiability problems have been studied in the literature ([Sin05],[ANOR09]). In this thesis four different encoding types are investigated: combinatorial encoding, sequential encoding, bit adder encoding, and cardinality networks.

The combinatorial encoding enumerates all possible undesired truth assignments and forbids them explicitly by generating corresponding clauses. While this approach may provide an efficient encoding for small cardinality constraints (for example $atMost_2(\{x_1, x_2, x_3\})$ would be encoded into the single clause $(\neg x_1 \vee \neg x_2 \vee \neg x_3)$), the number of generated clauses will grow exponentially with the number of variables. An alternative approach would be to explicitly enumerate all desired truth assignments.

The idea behind the sequential and bit adder encoding is to capture the sum of the considered variables and then forbid certain output values. For example, considering the assignment $(x_1, x_2, x_3) = (1, 1, 1)$, both encodings would calculate the sum 3, but the difference lies in the way how this number is encoded. The sequential encoding represents the sums as a unary number (number with base 1, e.g. $3_{10} = 111_1$), while the bit adder encoding represents the sum as a binary number (number with base 2, e.g. $3_{10} = 11_2$). The choice of the number representations has an impact on the number of generated

clauses, variables, and some other maxSAT properties. Clearly, by restricting certain outputs, the desired cardinality constraint can be captured.

Cardinality networks generate helper variables that are used to sort all the considered truth assignments and then insert clauses which forbid certain outputs. The sorting is performed in a similar way as a simple merge sort algorithm would work. For example, considering an assignment $(x_1 = 0, x_2 = 1, x_3 = 0, x_3 = 1)$, the helper variables a_{1-4} would represented the sorted version of this assignment $(a_1 = 1, a_2 = 1, a_3 = 0, a_4 = 0)$. Additional clauses are then inserted to forbid undesired assignments of the helper variables.

The performance of the different cardinality constraint encodings applied on two leading maxSAT solvers will be discussed in chapter 5.

3.3 Modeling of Hard Constraints

An employee cannot be assigned more than one shift on a single day. Since no employee should work in two shifts on the same day, it has to be ensured that no two variables $S_{i,d,t}$ and $S_{i,d,x}$ may be set to true at the same time if $t \neq x$ and $i \in I$, $d \in D$, $t, x \in T$ where I is the set of all employees, D is the set of all days in the scheduling horizon and T is the set of all possible shift types.

This constraint can be modeled with the use of an $atMost_1$ cardinality constraint.

$$atMost_1(\{S_{i,d,1}, S_{i,d,2}, ..., S_{i,d,|T|}\}) \qquad \forall i \in I, d \in D \tag{3.2}$$

Disallowed shift sequences. For the employee scheduling problem it is required that all employees need to rest for a minimum amount of time after they have worked in a shift. The length of the necessary rest period varies for each shift type. Because each shift has fixed starting and ending times during the day, the set of shift types that cannot follow a certain shift type t can be determined easily by considering all pairs of shift types and comparing their difference in start and ending times with a minimum rest time. The set of all shift types that are not allowed to follow a shift t is in the following referred to as R_t.

The constraints can also be thought of as a number of disallowed shift sequences which consist of two consecutive shifts and can be included in the partial weighted maxSAT formulation by inserting a clause for each sequence.

$$\bigwedge_{d=1}^{|D|-1} (S_{i,d,t} \rightarrow \neg S_{i,d+1,x}) \qquad \forall t \in T, x \in R_t \tag{3.3}$$

The maximum numbers of shifts for each type that can be assigned to an employee. In the employee scheduling problem some of the employees can have contracts which only allow them to work in specific shift types for a maximum number of

days. Such a limit could for example restrict the number of night shifts employees may work during their schedule to four, making any roster which assigns five night shifts to a single employee infeasible. The maximum numbers for each employee and shift type are given as parameters m_{it}^{max} with the problem instances, where $i \in I$ and $t \in T$.

Since this constraint can be seen as the basic case for a cardinality constraint, the detailed encoding into a Boolean satisfiability clauses is not further discussed here, but simply stated as an $atMost_{m_{it}^{max}}$ cardinality constraint instead:

$$atMost_{m_{it}^{max}}(\{S_{i,1,t}, S_{i,2,t}, ..., S_{i,|D|,t}\}) \qquad \forall i \in I, t \in T \tag{3.4}$$

Minimum and maximum working time. Each shift type assigns a certain amount of working time in minutes to its associated employees. Moreover the total number of the working time in minutes is restricted for each employee and must lie between a minimum and maximum bound. Those limits are given to the problem in form of the parameters b_i^{min} and b_i^{max} for each $i \in I$.

In order to formulate this constraint efficiently, additional helper variables that count the total number of minutes worked by an employee are introduced in the following. Their definition uses the shift lengths $l_t, \forall t \in T$ which are given as parameters to the problem and specify the number of working time in minutes required for shift t. Furthermore, their greatest common divisor is referred to as $g = gcd(l_t : t \in T)$. Consider for example three different shift types, with the first one lasting for 480, the second one lasting 620, and the third one lasting 120 minutes. In this case g would then be 20.

With g defined, simplified lengths sl_t can be calculated for all shifts $sl_t = \frac{l_t}{g}\ \forall t \in T$. Additionally, the maximum of the simplified shift lengths is defined as $sl_{max} = max\{sl_t : t \in T\}$.

Using these definitions the variable set U, which counts the units of time an employee i works on day d, can be introduced: For each employee and day, the helper variables $U_{i,d,x}, \forall i \in I, d \in D, x \in 1, ..., sl_{max}$ are generated. The following number of equivalences connects those helper variables with the decision variables:

$$S_{i,d,t} \leftrightarrow \bigwedge_{x=1}^{sl_t} U_{i,d,x} \wedge \bigwedge_{y=sl_t}^{sl_{max}} \neg U_{i,d,y} \tag{3.5}$$

Variables from the set U count the overall units of time that an employee works during the schedule. Therefore they can be used to set up two cardinality constraints that ensure the minimum and maximum working time constraint. Note that since simplified lengths are used for this calculation, the given limits have to be divided by the common divisor g and rounded appropriately:

$$atMost_{\lfloor b_i^{max}/g \rfloor}(\{U_{i,d,x} | d \in D, x \in \{1, ..., sl_{max}\}\}) \quad \forall i \in I \tag{3.6}$$

$$atLeast_{\lceil b_i^{min}/g \rceil}(\{U_{i,d,x} | d \in D, x \in \{1, ..., sl_{max}\}\}) \quad \forall i \in I \tag{3.7}$$

Maximum consecutive shifts. All employees are only allowed to work for a maximum number of consecutive days before they must have a day off. This maximum limit is given to the problem as parameter c_i^{max} for each $i \in I$. This constraint is included into the maxSAT formulation by introducing clauses that require a day off during all possible sequences of length c_i^{max}. Since this constraint assumes that the last day before the scheduling horizon sets a day off and the first day after the scheduling horizon also sets a day off, it is not necessary to consider any corner cases.

$$\bigvee_{x=0}^{c_i^{max}} X_{i,d+x} \qquad \forall i \in I, d \in \{1, ..., |D| - c_i^{max}\} \tag{3.8}$$

Minimum consecutive shifts. The considered variant of the employee scheduling problem requires that each employee works at least for a minimum of consecutive days. In other words there is a minimum for the number of consecutive shifts, which is given as parameter c_i^{min} for all $i \in I$, before an employee is allowed to have a day off.

Again corner cases do not have to be considered, since this constraint always assumes an infinite number of consecutive working days before and after the scheduling horizon. For all the other cases, the constraint is formulated into maxSAT clauses by implicating the minimum length shift sequence whenever a new shift sequence starts after a day off:

$$(X_{i,d} \wedge \neg X_{i,d+1}) \rightarrow (\bigwedge_{x=2}^{c_i^{min}} \neg X_{i,j+x}) \qquad \forall i \in I, d \in \{1, ..., |D| - 3\} \tag{3.9}$$

Minimum consecutive days off. This can be formulated similarly to the minimum consecutive shifts constraint. No corner cases have to be considered, as this constraint assumes an infinite sequence of days off before and after the scheduling horizon. The minimum limit of consecutive days off is given to the problem as parameter o_i^{min} for each employee $i \in I$.

Again a formulation variant which applies an implication of a minimum length day off sequence is used for this constraint, similar as described for the minimum consecutive shifts constraint.

$$(\neg X_{i,d} \wedge X_{i,d+1}) \rightarrow (\bigwedge_{x=2}^{o_i^{min}} X_{i,d+x}) \qquad \forall i \in I, d \in \{1, ..., |D| - 3\} \tag{3.10}$$

Maximum number of weekends. Whenever an employee has to work a shift on a Saturday or a Sunday in the schedule, the corresponding weekend is considered as a working weekend for this employee. The problem restricts the number of such working

weekends which is given as parameter a_i^{max} for each employee i. Because the scheduling horizon always starts on a Monday and ends on a Sunday, the number of weekends can be easily calculated as $w = \frac{|D|}{7}$. With w, additional helper variables $W_{i,x}$ can be introduced, that state whether or not an employee i works on the x-th weekend. The following equivalences are introduced to connect the W variables with the existing X variables in the maxSAT formulation. (Note that the x variables are multiplied with 7 in order to determine the day index of the x-th Sunday in the schedule).

$$W_{i,x} \leftrightarrow (\neg X_{i,(x\cdot 7)-1} \vee \neg X_{i,x\cdot 7}) \qquad \forall i \in I, x \in \{1, ..., w\} \tag{3.11}$$

With the help of those variables the following cardinality constraints can be declared in order to formulate the maximum number of weekends constraint:

$$atMost_{a_i^{max}}(\{W_{i,1}, W_{i,2}, ..., W_{i,w}\}) \qquad \forall i \in I \tag{3.12}$$

Days off. Employees may have certain days on which it is strictly required that they have a day off. Those are given to the problem as sets of day indices N_i for each employee i. Those day off constraints can be introduced in the maxSAT formulation by simply generating corresponding unit clauses:

$$X_{i,d} \qquad \forall i \in I, d \in N_i \tag{3.13}$$

3.3.1 Modeling of Soft Constraints

Requested shift types. Each employee may have some days where a certain shift type is requested for them to work in. Since this is not a hard constraint, a violation will be penalized with a given weight. The corresponding penalties are given to the problem as parameters $q_{i,d,t}$, where $i \in I$, $d \in D$ and $t \in T$. This constraint is included into the formulation with the insertion of simple weighted unit clauses for all the shift requests:

$$S_{i,d,t} \cdot q_{i,d,t} \qquad \forall (i,d,t) \quad where \quad \exists q_{i,d,t} \tag{3.14}$$

Unpreferred shift types. Similar to the requested shifts constraint, the problem may contain requests that require an employee to not work a particular shift on a certain day. The formulation is again based on weighted unit clauses depending on problem parameters $p_{i,d,t}$ that set the weight of an unpreferred shift, where $i \in I$, $d \in D$ and $t \in T$:

$$\neg S_{i,d,t} \cdot p_{i,d,t} \qquad \forall (i,d,t) \quad where \quad \exists p_{i,d,t} \tag{3.15}$$

Cover requirements. The number of employees that should be working in a shift type is defined for each day. This preferred value of working employees for shift t on day d is given to the problem in form of parameters u_{dt} for all $d \in D$ and $t \in T$. Furthermore for each of these values two penalty parameters v_{dt}^{min} and v_{dt}^{max} are used to penalize a possible under- or over-coverage of the preferred value.

Two cardinality constraints per cover requirement are introduced into the maxSAT formulation to handle this constraint. One for the over-coverage, which is penalized linearly depending on the weight v_{dt}^{max}, and another one for the under-coverage which is also penalized linearly depending on the weight v_{dt}^{min}:

$$atMost_{u_{dt}}(\{S_{1,d,t}, S_{2,d,t}, ..., S_{|I|,d,t}\}) \cdot v_{dt}^{max} \qquad \forall d \in D, t \in T \tag{3.16}$$

$$atLeast_{u_{dt}}(\{S_{1,d,t}; S_{2,d,t}; ...; S_{|I|,d,t}\}) \cdot v_{dt}^{min} \qquad \forall d \in D, t \in T \tag{3.17}$$

CHAPTER 4

A Hybrid Approach for the Employee Scheduling Problem

In the following chapter a new hybrid approach for solving the employee scheduling problem which combines heuristic methods together with exact solution techniques from constraint programming will be proposed. The problem representation as well as the problem's evaluation function are introduced in sections 4.1 and 4.2. The following sections then focus on techniques that are used during the local search part of the algorithm: Section 4.3 describes the used search neighborhoods and section 4.4 introduces the heuristic methods which are applied during the local search procedure. An exact solution approach using constraint programming is then described in section 4.5. Afterwards, a construction heuristic for the employee scheduling problem is proposed in section 4.6. Finally, in the last section 4.7 of this chapter, all proposed solution methods are combined within an iterated local search based procedure.

4.1 Problem Representation

The search techniques which are described in this chapter all use a direct representation of the scheduling horizon. The problem is represented by a simple $|I| \times |D|$ matrix where each row represents the schedule of an employee and each column stands for a day in the roster. Each cell in the matrix can either contain a single shift or be empty, which would assign either a certain shift type or a day off to the associated employee on the associated day. This representation ensures that the *"Employees can not be assigned more than one shift on a single day"* constraint is always fulfilled.

4.2 Evaluation of Solutions

Since a feasible solution must not violate any hard constraint, in [CQ14] it has been proposed to express the objective function f of a feasible solution S as follows (Assuming that variables x_{idt} are 1 if and only if employee i works in shift t on day d and otherwise 0, variables y_{dt} mark the total below the preferred cover for shift type t on day d, and variables z_{dt} mark the total above the preferred cover for shift type t on day d):

$$f(S) = \sum_{i \in I} \sum_{d \in D} \sum_{t \in T} q_{idt}(1 - x_{idt}) + \sum_{i \in I} \sum_{d \in D} \sum_{t \in T} p_{idt} x_{idt} + \sum_{d \in D} \sum_{t \in T} y_{dt} v_{dt}^{min} + \sum_{d \in D} \sum_{t \in T} z_{dt} v_{dt}^{max} \tag{4.1}$$

This evaluation function f calculates the objective value for feasible solutions appropriately to the problem definition and is therefore applied in all proposed algorithms of this chapter.

Additionally to having an objective value that measures the quality of feasible solutions, it is also desirable to compare the fitness of infeasible solution candidates during local search. For this purpose, the number of hard constraints violations can be considered. When capturing the sum of all such violations, each occurrence of all the mentioned hard constraints then counts as one violation, with the exception of the working time constraints and the maximum number of shifts per employee constraints, for which a potential violation is only counted once per employee.

In addition to counting just the number of violated hard constraints, the number of cells that are affected by the violations can also be considered. Measuring the cells that are involved with violations of hard constraints can have positive effects on local search, since it tries to iteratively improve a solution by the modification of cells. In some cases it might be necessary to perform several search steps until a constraint is resolved and while the number of violations might not change in between the necessary steps, an improvement can be visible through a reduced number of affected cells. The rules that determine which cells will get marked as affected by each constraint are described in section 4.4 of this chapter. Cells that are affected by a constraint violation are later also referred to as conflicting cells.

Assuming that v stands for the number of hard constraint violations and c stands for the number of cells which are affected by hard constraint violations, the following extended cost function g considers violated hard constraints as well as violated soft constraints when calculating the objective function for a candidate solution S:

$$g(S) = v \cdot M + c \cdot \alpha + f(S) \tag{4.2}$$

Note that M has to be chosen in such a way that it is greater than the highest possible value for $f(S)$, so that an infeasible solution can never have a lower cost than a feasible solution. α is a weight parameter that determines the influence of the number of conflicted

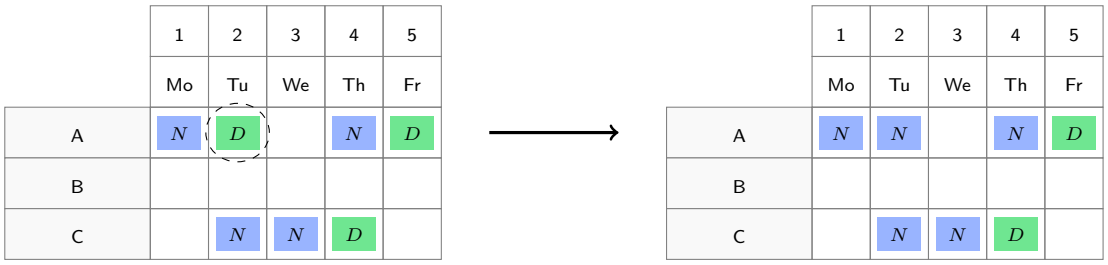

Figure 4.1: This figure shows an example of a shift change neighborhood move. The candidate solution presented on the left displays the schedule before the shift change. On the right hand side of the figure one can see the schedule after the day shift on Tuesday has been changed to a night shift for employee A.

cells. Also note that in case of a feasible candidate solution v and c will be both zero and $g(S)$ will therefore be equal to $f(S)$.

4.3 Search Neighborhoods for Employee Scheduling

In this section, the three different search neighborhoods (first introduced in [BCQB10]) that are used in the proposed local search approach are described. All of them differ in the way they make changes to a candidate solution, and work by either swapping or directly reassigning shifts in the roster. The considered neighborhoods which are described in this section are: Shift change, vertical swapping of cell assignments, and horizontal swapping of cell assignments.

In the following, all neighborhoods are first described as single shift neighborhoods. However, in addition to single shift moves also block neighborhoods will be applied in the proposed algorithm. The notion of block neighborhoods will be described later in this chapter.

4.3.1 Shift Change

The simplest neighborhood considers changes of single cells by inserting, modifying or deleting shift assignments during search. Such neighborhood moves can influence all of the problem's hard and soft constraints and are necessary in order to cover the whole search space. Figure 4.1 shows an illustration of a shift change move.

4.3.2 Vertical Swapping of Cells

During search the algorithm tries to reduce penalties that are caused from cover constraints violations by assigning appropriate shifts to each day in the schedule. At some point a situation might occur, where the cover requirements are fulfilled satisfyingly, but other constraints still cause violations. In such a scenario the vertical swap neighborhood, that

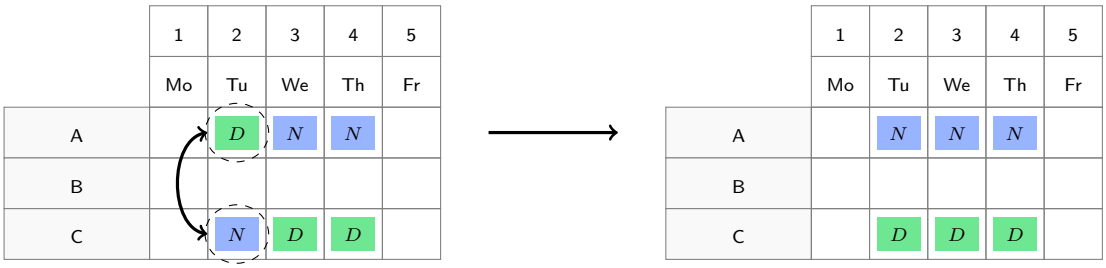

Figure 4.2: This figure shows a vertical cell swap. On the left hand side, one can see the schedule before the swap. The schedule after the swap shows that two shifts on Tuesday have been swapped and is displayed on the right side of the figure.

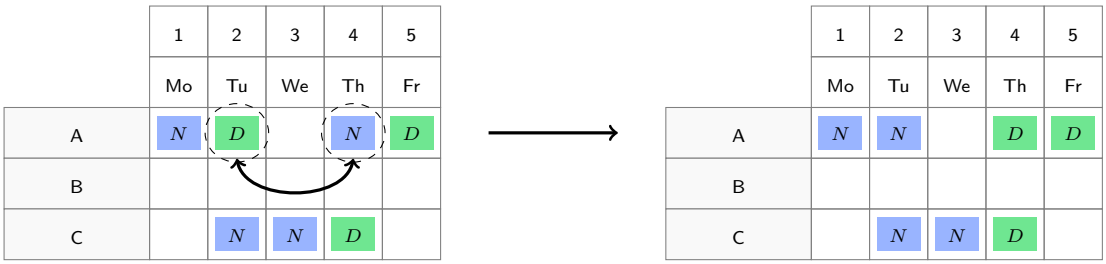

Figure 4.3: This figure shows a horizontal cell swap. The left side of the figure shows the schedule before the swap. On the right hand side one can see the schedule after the swap. In this case, the shift assignments on Tuesday and Thursday have been swapped for employee A.

tries to improve solutions by swapping two cell assignments within the same day, can be advantageous. By considering the vertical swap of two cells, neighborhood solutions will be explored that do not affect any cover requirement constraints. Figure 4.2 illustrates such a vertical cell swap.

4.3.3 Horizontal Swapping of Cells

The third possibility of generating neighborhood solutions is to swap cells within the schedule of a single employee. While the horizontal swapping of shifts might cause cover constraints violations, it can be useful to perform changes without violating other employee dependent constraints like the workload requirements or maximum shift requirements. An example for a horizontal swap is shown in Figure 4.3.

4.3.4 Block Neighborhoods

All of the three previously described neighborhoods can be extended to not only consider the modification of a solution by performing swaps with pairs of single cells or changing single cells in the schedule, but to also consider the modification of horizontal blocks

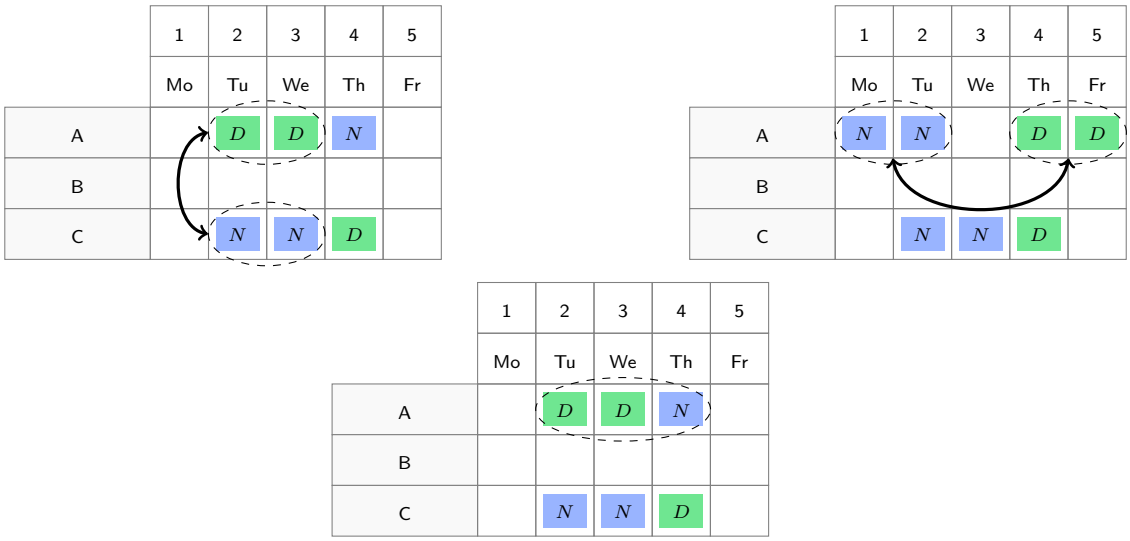

Figure 4.4: This figure shows examples for block move variants of the shift change, vertical cell swap, and horizontal cell swap neighborhoods. On the top left one can see vertical swap move with a block size of two. The schedule on the top right shows a horizontal swap, also with a block size of two. In the bottom of the figure the cell selection for a shift change with a block size of three is displayed. With such a shift change move, all of the selected cells could then for example be reassigned at the same time to contain a night shift.

of cells in one move. Instead of changing only single cell assignments, a block variant of the shift change neighborhood can then assign the same shift type or a day off to a number of consecutive cells within a single search step. Vertical and horizontal cell swap block moves work in similar way by swapping pairs of blocks that consist of consecutive cells lying in the schedule of the same employee. The length of the considered blocks can vary with each block move and is also referred to as block size. Examples for block neighborhood move variants of the three search neighborhoods are illustrated in Figure 4.4.

Although it is theoretically possible to cover the whole search space with just the single cell neighborhoods, there are reasons to include also block moves into local search as they can introduce a significant increase in performance. In [aH00] it is shown that cases can occur in which single cell neighborhoods alone are not able to reach certain solutions. For example if a solution wants to move a block of three consecutive shifts from one employee to another employee and there is a minimum consecutive shift constraint that requires sequences of at least three working shifts, a single shift neighborhood would need to perform three moves to reach the desired solution. However, in between those swaps the minimum consecutive shift constraint would be temporarily violated, which then will lead to a heavily increased objective value and local search therefore might not

allow the whole sequence of necessary search steps. However, with the consideration of vertical block swaps such a problem would not occur in the same situation.

A drawback that comes with the introduction of block neighborhoods is that more computing power is necessary in order to calculate and compare the outcomes of all possible neighborhood moves. Therefore, an upper limit for the considered block size should be chosen carefully, so that the number of possible moves is restricted. For the algorithms proposed in this chapter, the maximum length for consecutive shift sequences determined by the problem's constraints sets limit for the maximum block size. Single shift neighborhoods are still possible and can be thought of as block changes with a block size of one.

4.4 Local Search Methods for Employee Scheduling

In this chapter methods for the local search parts of the proposed hybrid solver are described. After the notion of conflicted cells has been introduced in 4.4.1, a variant of the min conflicts heuristic [MJPL92] which is applied during the main part of the search process is proposed in section 4.4.2. Additional methods which are used to explore large neighborhoods are then presented in section 4.4.3.

4.4.1 Conflicted Cells

In order to apply the min conflicts heuristic, that will be further described in section 4.4.2, the notion of conflicted cells has to be defined for each of the problem's constraints. In the following a list of all constraints and their in case of a violation corresponding set of conflicted cells is presented.

Disallowed shift sequences. Since this constraint forbids certain sequences of shift types with length two, in case of a violation there are two affected cells: The cell containing the shift assignment and the cell that contains its forbidden successor.

The maximum number of shifts for each type that can be assigned to an employee. If such a constraint is violated, then there are too many shifts of a certain type assigned to the same employee. All of the employee's cells that contain a shift of the corresponding type are therefore considered to be in conflict.

Minimum and maximum working time. If employees do not meet the minimum working time requirements than they have too many days off in their schedule. Therefore, all work free days are considered to be in conflict in case of such a violation. If on the contrary the maximum working time constraint is violated, all cells of the associated employee that contain shift assignments are considered to be in conflict.

Mo	Tu	We	Th	Fr	Sa	Su
*	*	*D*	*	*N*	*N*	*N*

Figure 4.5: This figure shows which cells are considered to be in conflict in case of a minimum consecutive shifts constraint violation. In this example the minimum of consecutive shifts is set to three and the asterisks indicate the cells that are considered to be in conflict. The length of the shift sequence containing only one day shift is too short, and therefore the two empty cells that lie before the shift are marked with an asterisk. Only one cell after the late shift is marked with an asterisk, since the following cells already contain shift assignments.

Maximum consecutive shifts. Whenever there occurs a sequence of consecutive shift assignments in the schedule of an employee that is longer as the allowed maximum, all cells that contain shift assignments within this sequence are considered to be in conflict.

Minimum consecutive shifts. If a candidate solution contains a sequence of shifts that has a length lower than the allowed minimum, those cells that come before and after the sequence and lie within the difference to the necessary minimum are considered to be in conflict. Cells that already have a shift assignment are not affected by this rule. Figure 4.5 shows an example of such a constraint violation and visualizes the corresponding cells which are in conflict.

Minimum consecutive days off. In case the minimum consecutive days off constraint is violated, the cells which are considered to be in conflict are determined in a similar way as already described for the minimum consecutive shifts constraint. Cells before and after the sequence lying within the difference to the necessary minimum are in conflict given that they contain a shift assignment.

Maximum number of weekends. If employees have more working weekends in their schedule than the allowed maximum, all weekend cells in their schedule that contain shift assignments are considered to be in conflict.

Days off. Day off requests are only defined for single shifts in the grid and therefore the corresponding cells will be considered to be in conflict in case of a violation.

Requested shift types. Similar to day off requests, shift requests affect single cells, which will be considered to be in conflict in case of a violation.

Unpreferred shift types. Similar to day off requests, unpreferred shift requests affect single cells, which will be considered to be in conflict in case of a violation.

Cover requirements. If a cover requirements constraints is violated, there are two cases that have to be distinguished: In case of an under coverage violation, all cells which do not contain a shift assignment will be considered to be in conflict on the corresponding day. On the other hand, in case of an over coverage violation, all cells that contain a shift assignment will be considered to be in conflict on the corresponding day.

4.4.2 The Min Conflicts Heuristic

The idea behind the min conflicts heuristic lies in selecting variables of a problem that are causing constraint violations and then focuses on the reduction of conflicts during search. The previous section defined the notion of conflicted cells and described which cells are considered to be in conflict in case of a constraint violation. Min conflicts chooses a conflicting cell randomly in each search step and then generates all possible neighborhood moves that include the selected cell. The heuristic then performs the move that leads to the lowest objective value and thereby tries to resolve as many violated constraints as possible as search progresses.

While focusing on only conflicted cells can often lead to fast improvements, a draw back of min conflicts is that the search can get stuck in local optima easily. Therefore, a variant of the min conflicts heuristic that makes use of a tabu list [Stü97] is described in this chapter. A tabu list is a data structure that keeps track of recently performed neighborhood moves during search and makes sure that equal moves will not be repeated in multiple search steps. If configured properly, such a tabu list can prevent any cyclic moves and local search will thereby be able to escape from local optimal solutions. Variants of min conflicts heuristics in combination with random walk and tabu search have been applied before in the domain of employees scheduling in [Mus06] and [BGM$^+$10].

The detailed min conflicts procedure which is used in the proposed hybrid approach for employee scheduling is defined in Algorithm 4.1.

An important property of the proposed min conflicts procedure is that hard constraint violations and soft constraint violations are considered in separate phases. In a first phase the algorithm tries to get rid of the hard constraints, and only if there are no hard constraint violations left, it considers conflicted cells that are associated to soft constraint violations.

The tabu list data structure plays also an important role in Algorithm 4.1. A potential neighborhood move is only accepted if no similar move is contained in the tabu list. However, there is an exception to this rule: If a move leads to a cost which is better than the best known solution, it will be in any case accepted.

4.4.3 Additional Exploration of the Neighborhood

In cases where min conflicts is not able to further improve a locally optimal solution but the search still has some time left, it can be beneficial to consider a larger of number of potential neighborhood moves in each search step. In this section the following

Algorithm 4.1: Min conflicts algorithm

```
Data: candidate solution, tabu list, bestCost
Result: candidate move
choose randomly a cell which is in conflict;
if candidate solution has hard constraint violations then
    choose a cell that is in conflict through a hard constraint violation;
else
    choose a cell that is in conflict through a soft constraint violation;
end
generate all possible moves using all three neighborhoods for selected cell;
bestMove ← find the best move which minimizes the total cost function;
bestMoveNotTabu ← find the best move which minimizes the total cost and is not
tabu;
if bestMove ≠ bestMoveNotTabu then
    if evaluate(bestMove) < bestCost then
        return bestMove;
    end
else
    return bestMoveNotTabu;
end
```

three algorithms are introduced to conduct an additional exploration of the neighborhood: Employee swap (Algorithm 4.2), employee improvement (Algorithm 4.3) and day improvement (Algorithm 4.4).

The employee swap procedure selects a single employee randomly and considers swapping the entire schedule assigned to this employee with the schedule from another employee. The procedure basically considers vertical swaps of a very large block size. Early experiments showed that the inclusion of such swaps can enhance search diversification.

The employee improvement and day improvement procedures use the same search neighborhoods as the min conflicts heuristic but do not restrict cell selection on only conflicted cells. Instead, both techniques consider all cells of the entire schedule from a single employee or day instead. After a day or employee has been selected randomly, they explore neighborhood moves for all cells which are associated to this day or employee. The employee improvement and day improvement procedures add two very large neighborhoods to the search that can consume a lot of runtime per iteration when dealing with larger instances.

4.5 Constraint Programming for Employee Scheduling

In this section an exact solution approach for the employee scheduling problem that uses CP [RvBW06] methods is described. Later in this chapter, the previously described local

Algorithm 4.2: Employee swap algorithm

```
Data: candidate solution, tabu list, bestCost
Result: candidate move
choose randomly an employee;
Generate all possible moves that swap the whole schedule of this employee with
another employee;
bestMove ← find the best move which minimizes the total cost function;
bestMoveNotTabu ← find the best move which minimizes the total cost and is not
tabu;
if bestMove ≠ bestMoveNotTabu then
    if evaluate(bestMove) < bestCost then
        return bestMove;
    end
else
    return bestMoveNotTabu;
end
```

Algorithm 4.3: Employee improvement algorithm

```
Data: candidate solution, tabu list, bestCost
Result: candidate move
choose randomly an employee and a second employee to swap with;
Generate all possible moves for all the cells of the first chosen employee using all
three neighborhoods, where the swap neighborhood is restricted to swaps with the
second chosen employee;
bestMove ← find the best move which minimizes the total cost function;
bestMoveNotTabu ← find the best move which minimizes the total cost and is not
tabu;
if bestMove ≠ bestMoveNotTabu then
    if evaluate(bestMove) < bestCost then
        return bestMove;
    end
else
    return bestMoveNotTabu;
end
```

Algorithm 4.4: Day improvement algorithm

Data: candidate solution, tabu list, bestCost
Result: candidate move

```
1  choose randomly a day in the schedule;
2  generate all possible moves for all the cells of the selected day using the vertical
   swap and shift change neighborhoods;
3  bestMove ← find the best move which minimizes the total cost function;
4  bestMoveNotTabu ← find the best move which minimizes the total cost and is not
   tabu;
5  if bestMove ≠ bestMoveNotTabu then
6  |  if evaluate(bestMove) < bestCost then
7  |  |  return bestMove;
8  |  end
9  else
10 |  return bestMoveNotTabu;
11 end
```

search techniques will be combined together with the methods from this section within an iterated local search based procedure in section 4.7.

In order to devise a CP approach for the employee scheduling problem, appropriate variables, value assignments and constraints have to be defined. Similar as with the previously described local search approach, a variable is generated for all the cells of the scheduling grid and their cell assignments should fulfill the problem's hard- and soft constraints. Possible variable values are all the different shift types that are given with a problem as well as a day off assignment. The goal then is to find a complete variable assignment that builds an optimal feasible solution (i.e. a solution with lowest possible cost that does not violate any of the problem's hard constraints). In order to reach such a solution, usually a backtracking based tree search [RvBW06] is conducted. Two essential techniques play a role when using a constraint programming based solver: Enumeration and constraint propagation. Enumeration techniques on the one hand provide efficient ways to search through all possible combinations of variable assignments and constraint propagation techniques on the other hand provide options to intelligently look ahead during search and to prune the search space by eliminating groups of infeasible variable assignments. Since the employee scheduling problem is an optimization problem, branch and bound mechanisms can also be included into the search procedure. In the following sections the application of those techniques regarding the employee scheduling problem is described.

4.5.1 Enumeration Strategies

The use of an appropriate variable and value ordering is essential for any CP based solving method. Good orderings have significant impact on the performance of pruning

techniques and how they can remove search branches to reduce the problem's search space.

Most of the problem's hard constraints focus on the schedule of an employee. For this reason, it is beneficial to order the variables in such a way that a tree search will assign the shifts of a single employee's schedule one after the other. In other words, the first cell selected by search will be the first day of the first employee's schedule, continued by the second cell of the first employee's schedule and so on. The first cell of the second employee's schedule will only be considered as soon as the schedule of employee one has been completely determined. With such an ordering, constraint propagation will be able to detect many hard constraint violations early in the search. Regarding the value selection, the constraint programming method proposed in this chapter simply uses the ordering of the shift types as given in the problem description (Day off assignment are always considered last).

4.5.2 Constraint Propagation

In the algorithm used in this chapter forward checking [RvBW06] is applied as a look ahead mechanism during tree search. This technique checks after each variable assignment if it is still possible to satisfy all hard constraints considering the remaining unassigned variables and their domains. It then removes all values from variable domains that would violate any hard constraint. For example, if four consecutive shifts have already been assigned in the schedule of an employee and only four consecutive shifts are allowed, all shift assignments will be removed from the domain of the following cell. If any variable has an empty domain after this procedure, the algorithm will exit the current search branch and backtrack to the last variable assignment that has been performed.

4.5.3 Branch and Bound

Since the employee scheduling problem is an optimization problem a branch and bound [LD10] strategy can be included in the algorithm. Whenever a leaf of the search tree is reached, the objective value of the corresponding solution is then stored as an upper bound for the remaining search space. Between each following variable assignment the lower bound for the cost of the current partial assignment is then calculated and compared to the upper bound. If the lower bound is greater or equal than the upper bound, the search can prune the current branch and backtrack until the lower bound is again lower than the upper bound.

Calculation of lower Bounds

The lower bound set by a partial solution must never overestimate the costs that are reachable from the partial assignment but may underestimate it. Therefore, costs produced by violated shift requests and unpreferred shift constraint violations are only taken into account during calculation of this bound if the partial assignment already violates such a constraint. Regarding the cover requirement constraints on the other

hand the calculation also considers the lowest possible cost increase that could occur through missing assignments. Given a partial assignment, the number of already assigned shifts is taken and used to include any penalties that can arise from over coverage constraints. Regarding the under coverage constraints, for each day the number of unassigned variables are considered as potential shift assignments. Since for the considered employee scheduling problem instances all different shift types have the same penalties in their cover requirements, the assumed under coverage can be distributed over under coverage constraints for all shift types.

The overall constraint programming based procedure which utilizes all of the mentioned techniques is described in Algorithm 4.5.

Algorithm 4.5: CP search algorithm

```
   Data: solution, bestUpperBound, unassignedVariables
   Result: bestSolution
 1 if unassignedVariables is empty then
 2 |   return solution;
 3 end
 4 nextVariable ← getNextVariable(unassignedVariables);
 5 while domain(nextVariable) is not empty do
 6 |   nextVariable.value ← getNextValue(domain(nextVariable));
 7 |   do constraint propagation using the newly assigned variable;
 8 |   lowerBound ← calculateLowerBound(solution);
 9 |   if lowerBound < bestUpperBound then
10 |   |   newSolution ← doCPSearch(solution, bestUpperBound,
   |   |   unassignedVariables);
11 |   |   newUpperBound ← evaluate(newSolution);
12 |   |   if newUpperBound < bestUpperBound then
13 |   |   |   bestUpperBound ← newUpperBound;
14 |   |   |   bestSolution ← newSolution;
15 |   |   end
16 |   end
17 |   unassignVariable(nextVariable);
18 end
19 return bestSolution;
```

4.6 A Construction Heuristic for Employee Scheduling

With the use of the local search techniques that are proposed in this chapter it is possible to find solutions even using an empty initial solution. The shift change neighborhood can then assign shift types to empty cells and thereby generate a feasible schedule in the process. While this method often works well if applied on smaller instances, some of the

instances have very large schedules, and precious time might be lost if search has to start with an empty schedule. For this reason, a construction heuristic that greedily generates an initial schedule can have improving effects on search and will be introduced in the following.

The main goal of the proposed construction heuristic is to generate an initial solution that is feasible or includes at least a low number of hard constraint violations. Shift requests, unpreferred shift requests and cover requirements are completely ignored, as they can only cause soft constraint violations.

The heuristic generates a solution by assigning shifts to cells in the scheduling grid one after the other, starting with day one of the first employee and ending with the last day of the last employee in the schedule. In the process it tries to consider all hard constraints by predicting the occurrence of forced assignments. A forced day off assignment for example could be the result of different hard constraints (e.g. day off requirement, maximum shift constraint) together with an already given partial assignment. On the contrary, at some points it might also be necessary to assign a certain shift type to a cell (e.g. through the minimum consecutive shifts or minimum working time constraints). The heuristic tries to predict whatever assignment will be necessary, but sometimes it has to guess what assignment should be made. In such a case a certain shift or day off assignment is picked randomly, although the likelihoods of the different options are still influenced based on the number of days left to assign and weighted against the minimum working time constraint. For example if the minimum working time constraint requires two additional shifts and only two cells are left to assign, a day off will not be considered as an option.

The detailed procedure for the construction heuristic is described in Algorithm 4.6.

Algorithm 4.6: Construction heuristic

```
   Result: initialSolution
1  solution ← generate initial empty solution for the problem;
2  for each employee do
3      lowestShiftLength ← calculate the lowest possible shiftlength;
4      increase the allowed weekend limit by four to reduce the strictness of the max
       weekend constraint;
5      for each day in the schedule do
6          calculate the difference to the next forced day off;
7          determine if day off is allowed for the current day;
8          determine how many shift assignments are still necessary based on
           lowestShiftLength and minimum working time constraint;
9          select the assignment for the current employee and day based on the
           gathered information and update solution;
10     end
11 end
12 return solution
```

4.7 An Iterated Local Search for Employee Scheduling

Although local search alone often can produce satisfying results, in the literature ([LMS01]) it has been shown that significant performance improvements can be achieved by using an iterated local search based procedure. The main idea behind iterated local search is to iteratively call an embedded local search, and to thereby perform a greater exploration of the search space. After each iteration, the currently best known solution is perturbed to provide a good starting point for the next run of the embedded local search.

In this section an iterated search procedure for employee scheduling is proposed in order to combine all of the methods that have been previously introduced in this chapter. The min conflicts heuristic as well as the additional local search methods serve as the embedded local search and the construction heuristic is used to generate an initial solution. Finally, a variant of the constraint programming based algorithm is used to perturb solutions after each run of the embedded local search.

4.7.1 The Embedded Local Search

As already mentioned, the embedded local search combines the previously introduced min conflicts algorithm as well as the employee swap, employee improvement, and day improvement procedures. In the following some details about how these techniques are utilized within iterated local search are given.

The min conflicts heuristic can often locate good neighborhood moves fast but has the drawback of easily being stuck in local optima, while the other three procedures take a deeper look on the neighborhood, but suffer from a high runtime that is consumed during each iteration. Therefore, the embedded local search is separated into a two phase approach. In the first phase local search tries to get large improvements quickly by applying only the min conflicts based procedure to generate neighborhood moves. If no improvement can be made using only min conflicts within a predefined limit that is relative to the problem's instance size, local search will enter a second phase. In this second phase all procedures (including min conflicts) will then be considered when selecting good neighborhood moves. In order to reduce the time consumed in each iteration, the processing of those neighborhoods is ordered, and a search move will be immediately accepted if it yields an improvement to the currently best known solution without considering the other neighborhoods.

4.7.2 Accepting Potential Neighborhood Moves

As soon as the best neighborhood move is determined, it still has to be determined if the move really should be accepted or not. If the move brings an improvement over the current solution it will be accepted in any case, however if it would increase the current cost, an acceptance mechanism which is based on the quality of the potential moves is applied. This mechanism which decides whether or not a move should be accepted uses

ideas from the simulated annealing meta heuristic [Kir84] and calculates an acceptance probability based on the change in quality.

The following equation shows how the probability to accept moves is calculated, assuming that h_{new} and $h_{current}$ are the number of hard constraint violations of the candidate and the current solution, s_{new} and $s_{current}$ are the soft constraint costs, and c_{new} and $c_{current}$ are the number of conflicts caused by hard constraint violations. Regarding the $costChange$ parameter, $costChange_1$ will be used to calculate the acceptance probability if the current solution is feasible. If on the other hand the current solution is infeasible and still contains hard constraint violations, $costChange_2$ will be used to calculate the acceptance probability. M is set to a value that is large enough so that improvements in the soft constraint penalties can never cause an increase in the number of hard constraint violations. The β parameter is used to reduce the influence of the number of cell conflicts in such a way, that it does not outweigh the number of hard constraint violations.

$$\begin{aligned} costChange_1 &= (h_{new} - h_{current}) \cdot M + s_{new} - s_{current} \\ costChange_2 &= h_{new} - h_{current} + (c_{new} - c_{current})/\beta \\ acceptanceProbability &= e^{-costChange_{1/2}} \end{aligned} \tag{4.3}$$

4.7.3 Perturbing the Current Solution

As soon as the embedded local search cannot find any new improvements within a given iteration limit, the iterated local search procedure enters its perturbation phase.

The goal of the perturbation phase is to modify the best solution found by local search in such a way that it can be used as a good new starting point for the next iteration. In the algorithm proposed in this chapter two different perturbation mechanisms are applied, whose selection depends on whether the current solution is feasible or not. If the best solution produced by the embedded local search is infeasible, the perturbation phase will simply restart by using the construction heuristic on an empty schedule. If the solution produced by local search on the other hand is feasible, parts of its cells will be emptied and a variant of the proposed constraint programming method will try to improve the resulting partial solution.

This constraint programming based procedure will then try to find the optimal solution within predefined time limit of ten seconds. The best feasible solution that can be found by this method will then be used directly as the starting point for the following local search. If however no feasible solution can be found within the time limit or if no feasible solution regarding the partial assignment exists at all, the solution with all feasible cell assignments that could be generated will be used as new starting point.

Regarding the constraint programming based perturbation it has to be noted, that only a part of the input solution's cells will be emptied and thereby are assignable through the perturbation mechanism. The remaining cells are considered as fixed during the whole

constraint programming based procedure. How those cells are selected has a large impact on the perturbation and can be seen in Algorithm 4.7.

Algorithm 4.7: Select cells for CP search algorithm

```
   Data: currentSolution
   Result: currentSolution with partially emptied cells
 1 if currentSolution contains hard constraint violations then
 2 |   for each employee do
 3 |   |   if the employee's schedule contains hard constraint violation
   |   |   ∨resetProbability <= random() then
 4 |   |   |   empty total schedule of this employee;
 5 |   |   end
 6 |   end
 7 else
 8 |   for each day do
 9 |   |   if the day has unfulfilled cover requirements
   |   |   ∨resetProbability <= random() then
10 |   |   |   empty all cells of this day;
11 |   |   end
12 |   end
13 end
14 return currentSolution;
```

Finally the complete iterated local search based procedure that forms the main method of the hybrid algorithm that has been proposed in this chapter is described in Algorithm 4.8.

Algorithm 4.8: Iterated Local Search

```
   Data: problem description, time limit, iteration limit, min conflicts limit
   Result: candidate move
 1 initialize tabu list;
 2 currentSolution ← constructionHeuristic();
 3 bestSolution ← currentSolution;
 4 overallBestSolution ← currentSolution;
 5 iterationCount ← 1;
 6 minConflictCount ← 1;
 7 minConflictPhase ← true;
 8 while ¬ out of time do
 9     while iterationCount < iteration limit ∧¬ out of time do
10         bestMove ← minConflicts(currentSolution, tabu list, cost(bestSolution));
11         if cost(bestMove) > cost(bestSolution) ∧(¬minConflictsPhase ∨ iterationCount >
           min conflicts limit) then
12             minConflictsPhase ← false;
13             candidateMove ← employeeSwap(currentSolution, tabu list, cost(bestSolution));
14             if cost(candidateMove) < cost(bestMove) then bestMove ← candidateMove;
15             if cost(bestMove) > cost(bestSolution) then
16                 candidateMove ← improveEmployee(currentSolution, tabu list,
                   cost(bestSolution));
17                 if cost(candidateMove) < cost(bestMove) then bestMove ← candidateMove;
18                 if cost(bestMove) > cost(bestSolution) then
19                     candidateMove ← improveDay(currentSolution, tabu list,
                       cost(bestSolution));
20                     if cost(candidateMove) < cost(bestMove) then
                       bestMove ← candidateMove;
21                 end
22             end
23         end
24         if acceptanceProbability(cost(bestMove), cost(currentSolution)) >= random() then
25             currentSolution ← doMove(bestMove);
26             update tabu list;
27             if cost(currentSolution) < cost(bestSolution) then
28                 bestSolution ← currentSolution;
29                 iterationCount ← 1;
30             end
31         end
32         iterationCount ← iterationCount + 1;
33     end
34     update overallBestSolution;
35     if bestSolution contains hard constraint violations then
36         currentSolution ← constructionHeuristic();
37     else
38         emptyConflictedCellsForCp(currentSolution);
39         currentSolution ← perturbeSolutionUsingCp(currentSolution);
40     end
41     bestSolution ← currentSolution;
42     update overallBestSolution;
43     reset tabu list;
44 end
45 return overallBestSolution;
```

CHAPTER 5

Experimental Evaluation

This chapter presents the experimental environment that was used to evaluate the proposed approaches for employee scheduling and provides a detailed report on the results.

Section 5.1 introduces the experimental environment that was used and gives a detailed presentation of the considered problem instances.

In section 5.2 the experiments that were conducted using the maxSAT model with different cardinality constraints and maxSAT solvers will be described. Furthermore, results will be reported and compared to existing approaches that use integer programming.

Section 5.3 describes the experiments that were used to evaluate the proposed hybrid approach for the employee scheduling problem. The results that were gathered by the hybrid algorithm are listed and compared with results produced by state of the art methods which include heuristic as well as exact solution techniques.

5.1 Experimental Environment

All of the experiments that are mentioned in this chapter were conducted using the 24 problem instances that are described in [CQ14]. According to the authors, those instances were designed to reflect real life scheduling problems that are challenging but still intuitive to understand. All of the instances vary in their size and the included scheduling periods range from one week up to one year, requiring up to 180 employees and 32 shift types that have to be assigned. Table 5.1 lists all of the considered problem instances in detail.

If not noted otherwise all of the experiments were run on an Intel Xeon E5345 2.33GHz machine that has a total of 48GB RAM.

Table 5.1: Instances from [CQ14]. Lower bounds were obtained using the integer programming model and the Gurobi solver. Upper bounds for instances 19-24 were obtained by the hybrid approach proposed in this thesis. Results formatted in bold face denote proven optimal solutions.

Instance	Weeks	Employees	Shift types	Lower bound	Upper Bound
Instance1	2	8	1	**607**	**607**
Instance2	2	14	2	**828**	**828**
Instance3	2	20	3	**1001**	**1001**
Instance4	4	10	2	**1716**	**1716**
Instance5	4	16	2	**1143**	**1143**
Instance6	4	18	3	**1950**	**1950**
Instance7	4	20	3	**1056**	**1056**
Instance8	4	30	4	1297	1308
Instance9	4	36	4	406	439
Instance10	4	40	5	**4631**	**4631**
Instance11	4	50	6	**3443**	**3443**
Instance12	4	60	10	**4040**	**4040**
Instance13	4	120	18	1346	1486
Instance14	6	32	4	1277	1280
Instance15	6	45	6	3806	4378
Instance16	8	20	3	3224	3225
Instance17	8	32	4	5726	5851
Instance18	12	22	3	4351	4760
Instance19	12	40	5	2945	3688
Instance20	26	50	6	4743	5936
Instance21	26	100	8	20868	22020
Instance22	52	50	10	24064	37912
Instance23	52	100	16	2765	41574
Instance24	52	150	32	?	54344

5.2 Experimental Evaluation of the MaxSAT Model

A large number of experiments with generated maxSAT encodings for the 24 instances described in [CQ14] have been conducted. The encoded maxSAT instances are available online in DIMACS format and can be downloaded at [1].

In the processed benchmarks two maxSAT solvers which performed well on timetabling instances in the maxSAT evaluation 2015 [max15] have been used: WPM3 [ADG15] and Optiriss using the default configuration. The latter uses the riss framework [KKMS15] in combination with the publicly available OpenWBO solver [MML14]. Both solvers were ranked first and second in the industrial category for partial weighted maxSAT problems. Besides being the leaders in their category, both solvers have also shown to provide good results for high school timetabling and timetabling instances, which share similarities with the considered employee scheduling problem.

5.2.1 Comparison of different Cardinality Constraint Encodings

Because the proposed model utilizes a number of cardinality constraints, a crucial point in the configuration of the experiments turned out to be the determination of which cardinality constraint encodings should be used in order to get good results with the maxSAT solvers. There are five constraints which are affected in the maxSAT formulation: The *cover requirement* constraint, the *workload requirement* constraint, the *maximum number of shifts* constraint, the *maximum number of weekends* constraint, and the *one shift per day* constraint. For those, four different encoding variants were applied: *combinatorial encoding*, *sequential encoding*, *cardinality networks encoding*, and *bit adder encoding*. An implementation from [DM14] was used to encode those constraints.

If all possible combinations for encoding the cardinality constraints in the model would have been considered, a total amount of $4^5 = 1024$ different encodings would have to be generated for each problem instance. In order to reduce this large amount of possibilities, the number of generated variables and clauses for all constraint/encoding pairs were investigated at first in order to gather a first insight on their importance. The results can be seen for one instance in Table 5.2.

The combinatorial encoding turned out to be impractical in most cases and often it was not even possible to generate maxSAT encodings. The huge amount of produced clauses required by this cardinality constraint encoding forced the model generator to run out of memory when dealing with larger instances. When looking at the numbers displayed in Table 5.2, it can be seen that the *maximum number of weekends* constraint and the *one shift per day* constraint have a relatively low impact compared to the other constraints. As this behavior appeared also with other instances, in the next step only the sequential encoding has been considered regarding the *maximum number of weekends* constraint and only the combinatorial encoding has been considered regarding the *one shift per day* constraint in the remainder of the experiments. With the elimination of the combinatorial

[1] http://www.dbai.tuwien.ac.at/research/project/arte/maxsat_employeescheduling/

Table 5.2: Overview on the number of generated variables (vars.) as well as the hard- and soft-clauses (h.c. and s.c.) for all the cardinality constraint/encoding pairs for instance five.

		Combinatorial	Sequential	Cardinality N.	Bit adders
Cover Req.	vars.	10192	7616	7056	5096
	h.c.	35616	28672	21168	17808
	s.c.	896	896	896	896
Workload Req.	vars.	Out of memory	6176	8032	5088
	h.c.	Out of memory	20240	21760	14864
	s.c.	Out of memory	0	0	0
Max shifts	vars.	Out of memory	3010	3520	6374
	h.c.	Out of memory	11928	10658	22646
	s.c.	Out of memory	0	0	0
Max weekends	vars.	0	124	160	176
	h.c.	46	420	496	602
	s.c.	0	0	0	0
One shift per day	vars.	0	896	896	1344
	h.c.	448	2688	3136	4480
	s.c.	0	0	0	0

encoding in the configuration options because of the caused inconveniences with larger instances and only three constraints remaining, only $3^3 = 27$ possible combinations of cardinality constraint configurations are left to examine.

In order to determine the best configuration for both WPM3 and Optiriss, nine instances of different sizes were selected and experiments with all of the 27 possible encoding variants were conducted under a time limit of 30 minutes. The results of those experiments can be seen in Table 5.3 and Table 5.4 for Optiriss and WPM3 respectively.

A comparison of those results reveals that there is no general best combination of cardinality constraint encodings and good encodings are highly dependent on the solver which is used. While Optiriss prefers the adder encoding for the *cover requirements* constraint, the sequential encoding shows the best results for WPM3. The best candidates for each solver were selected by considering the sums of the results over all instances for each combination of cardinality encodings. Finally, the encodings which led to the minimum of all those sums were taken to generate the instances for the final experiments. Therefore, the combinations of cardinality constraint encodings that were used for Optiriss are as follows: bit adder encoding for the *cover requirements* constraint, cardinality networks for the *workload requirements* constraint, and the sequential encoding for the *maximum number of shifts* constraint. The combinations of cardinality constraint encodings for WPM3 on the other hand are: The sequential encoding for the *cover requirements* constraint, the *workload requirements* constraint, and the encoding which uses cardinality networks for the *maximum number of shifts* constraint.

5.2.2 Final Experiments and Comparison of Solvers

By using the encodings mentioned above, maxSAT instances for the original problems 1-21 were created. Although the formulation can be used to encode Instances 22-24,

Table 5.3: Best results found by Optiriss using different combinations of cardinality encodings. The first column describes the cardinality encodings used for the *cover requirement/workload requirement/maximum number of shifts* constraints. Encoding names have been abbreviated: seq. = sequential encoding, card. = cardinality networks, adder = bit adders. In each column the best result is formatted in boldface.

Optiriss	**Best solutions found in 30 minutes time limit**				
Cardinality encoding	**Inst. 2**	**Inst. 4**	**Inst. 7**	**Inst. 9**	**Inst. 11**
seq./seq./seq.	837	5626	13333	12655	40435
seq./seq./card.	837	5626	12000	11659	40435
seq./seq./adder	839	5122	10078	11533	23720
seq./card./seq.	840	6002	15318	12460	32768
seq./card./card.	840	6002	12111	12242	32768
seq./card./adder	838	5215	11474	12758	24905
seq./adder/seq.	841	5407	14319	11044	34612
seq./adder/card.	841	5407	15148	11662	34612
seq./adder/adder	840	5331	11785	12752	25633
card./seq./seq.	841	5609	13813	14353	32281
card./seq./card.	841	5609	15211	11423	32281
card./seq./adder	**834**	5723	11987	13250	25631
card./card./seq.	**834**	6210	14080	12156	37028
card./card./card.	**834**	6210	13779	13154	37028
card./card./adder	841	5316	10682	10641	22130
card./adder/seq.	837	5711	13002	12570	32618
card./adder/card.	837	5711	13492	12785	32618
card./adder/adder	838	5504	9689	12976	24844
adder/seq./seq.	844	3900	5762	7729	15916
adder/seq./card.	844	3900	5741	7526	15916
adder/seq./adder	852	3720	5228	7437	16624
adder/card./seq.	853	**3608**	5421	**6394**	**15420**
adder/card./card.	853	**3608**	5852	6804	**15420**
adder/card./adder	847	3918	5452	7239	16464
adder/adder/seq.	845	3907	5411	7716	16627
adder/adder/card.	845	3907	5746	7422	16627
adder/adder/adder	850	3798	**5040**	7215	16436
Optiriss	**Best solutions found in 30 minutes time limit**				
Cardinality encoding	**Inst. 12**	**Inst. 14**	**Inst. 16**	**Inst. 18**	
seq./seq./seq.	57680	17959	15584	35073	
seq./seq./card.	58369	16665	15584	39555	
seq./seq./adder	34964	17549	13263	25829	
seq./card./seq.	57575	16761	15635	37084	
seq./card./card.	54138	16630	15635	37641	
seq./card./adder	33939	18362	14544	23932	
seq./adder/seq.	61229	17454	16013	34284	
seq./adder/card.	52854	16043	16013	28074	
seq./adder/adder	36632	15555	13937	27604	
card./seq./seq.	72062	19358	16093	37501	
card./seq./card.	49699	18247	16093	38147	
card./seq./adder	32074	17934	14776	28188	
card./card./seq.	56279	19044	16903	37778	
card./card./card.	50404	15546	16903	35638	
card./card./adder	32239	16918	14880	26855	
card./adder/seq.	62154	18980	17419	38269	
card./adder/card.	49096	18565	17419	30601	
card./adder/adder	33340	18593	15990	29781	
adder/seq./seq.	28602	10076	12546	21039	
adder/seq./card.	31000	9875	12546	22548	
adder/seq./adder	28694	**8777**	12223	21095	
adder/card./seq.	28598	9776	13026	20710	
adder/card./card.	30324	9144	13026	**20225**	
adder/card./adder	29596	9555	13049	20280	
adder/adder/seq.	**27193**	9758	11939	20462	
adder/adder/card.	29606	9931	11939	20504	
adder/adder/adder	29417	9756	**11707**	20996	

Table 5.4: Best results found by WPM3 using different combinations of cardinality encodings. The first column describes the cardinality encodings used for the *cover requirement/workload requirement/maximum number of shifts* constraints. Encoding names have been abbreviated: seq. = sequential encoding, card. = cardinality networks, adder = bit adders. In each column the best result is formatted in boldface.

WPM3	**Best solution found in 30 minutes time limit**				
Cardinality encoding	**Inst. 2**	**Inst. 4**	**Inst. 7**	**Inst. 9**	**Inst. 11**
seq./seq./seq.	**828**	3189	5510	10631	12183
seq./seq./card.	**828**	3189	**4596**	10949	12183
seq./seq./adder	**828**	3494	8959	10248	23420
seq./card./seq.	**828**	3090	7446	11132	**11516**
seq./card./card.	**828**	3090	6545	11405	**11516**
seq./card./adder	**828**	**2688**	8351	12154	24114
seq./adder/seq.	**828**	2784	7712	12178	12478
seq./adder/card.	**828**	2784	8553	**10033**	12478
seq./adder/adder	**828**	2893	9364	10964	24195
card./seq./seq.	835	3394	5230	10605	17224
card./seq./card.	835	3394	6815	11037	17224
card./seq./adder	**828**	4082	7562	11062	25444
card./card./seq.	**828**	3087	7143	10240	13888
card./card./card.	**828**	3087	8147	10942	13888
card./card./adder	839	3704	9670	10531	25626
card./adder/seq.	840	3695	7543	11871	15393
card./adder/card.	840	3695	7760	11235	15393
card./adder/adder	**828**	3103	9287	12374	22719
adder/seq./seq.	1550	3718	10502	13982	29673
adder/seq./card.	1550	3718	11315	12780	29673
adder/seq./adder	1159	3198	9478	14674	26133
adder/card./seq.	1563	3994	9365	11256	31595
adder/card./card.	1563	3994	9253	12773	31595
adder/card./adder	856	4212	9791	12693	25827
adder/adder/seq.	1469	4108	10292	10771	29083
adder/adder/card.	1469	4108	9476	11963	29083
adder/adder/adder	1359	3702	10100	10935	26467
WPM3	**Best solution found in 30 minutes time limit**				
Cardinality encoding	**Inst. 12**	**Inst. 14**	**Inst. 16**	**Inst. 18**	
seq./seq./seq.	23937	18045	**10292**	19771	
seq./seq./card.	**18770**	16303	**10292**	18498	
seq./seq./adder	1697590	15297	12738	21408	
seq./card./seq.	22010	15419	12528	19191	
seq./card./card.	19845	16285	12528	19241	
seq./card./adder	1697590	16654	16099	22100	
seq./adder/seq.	22536	17130	12550	**17277**	
seq./adder/card.	22734	16330	12550	20139	
seq./adder/adder	1697590	15155	15031	20793	
card./seq./seq.	24142	18272	12015	21095	
card./seq./card.	23726	18948	12015	22605	
card./seq./adder	32150	18455	14126	29910	
card./card./seq.	24206	16321	12848	25567	
card./card./card.	23716	16864	12848	21097	
card./card./adder	1697590	**14915**	16176	24417	
card./adder/seq	25331	17055	13360	24620	
card./adder/card.	26272	18104	13360	25051	
card./adder/adder	1697590	17490	16998	30144	
adder/seq./seq.	48422	18356	18064	31426	
adder/seq./card.	44948	19731	18064	29955	
adder/seq./adder	38822	18376	16497	27336	
adder/card./seq.	42744	19131	16259	28860	
adder/card./card.	44272	18959	16259	31694	
adder/card./adder	37582	18494	16343	30929	
adder/adder/seq.	42648	15723	17590	31791	
adder/adder/card.	45583	20184	17590	27403	
adder/adder/adder	35857	18143	18593	29081	

unfortunately maxSAT instances for those three problems could not be generated, since the used generator ran out of memory due to their large size (about 20 GB). The final experiments were conducted using both solvers, giving them a time limit of four hours for each of the 21 instances. The results of those benchmark tests can be seen in Table 5.5.

Table 5.5: The final results obtained for Instance 1-21 using WPM3 and Optiriss, using the selected cardinality constraint encodings described in this thesis. For comparison, the best known solutions using the exact methods described in [CQ14] are also included. Results formatted in bold face denote proven optimal solutions.

Instance	WPM3	Optiriss	Branch and Price [CQ14]	Gurobi [CQ14]
Instance 1	**607**	**607**	**607**	**607**
Instance 2	**828**	835	**828**	**828**
Instance 3	1009	3475	**1001**	**1001**
Instance 4	3102	3608	**1716**	**1716**
Instance 5	4037	3645	1160	**1143**
Instance 6	6150	6941	1952	**1950**
Instance 7	4596	5421	1058	**1056**
Instance 8	11018	7617	1308	1323
Instance 9	10949	6394	439	439
Instance 10	16435	15350	**4631**	**4631**
Instance 11	12183	15420	**3443**	**3443**
Instance 12	18770	28598	4046	**4040**
Instance 13	6110163	69203	-	3109
Instance 14	16303	9776	-	1280
Instance 15	30833	16506	-	4964
Instance 16	10292	13026	3323	3233
Instance 17	22002	22073	-	5851
Instance 18	18498	14433	-	4760
Instance 19	1698538	50274	-	5420
Instance 20	5519316	147325	-	-
Instance 21	14715064	-	-	-

When comparing the outcomes for WPM3 and Optiriss, one finds that is not possible to point out a clear winner which performs better over all the instances. While WPM3 performs significantly better on the smaller instances (Instances 1-7 and 11-12), it does not produce good solutions for the larger instances (Instances 8-10 and 13-21). Using Optiriss provides better results when it comes to solving the larger instances, except for the last three instances where the solver could not find any solution under four hours.

When comparing the proposed approach with another existing exact method based on integer programming, which was provided in [CQ14] (last two columns in the table), it can be concluded that both maxSAT solvers could not find new unknown optimal results. However they could provide optimal solutions for instances 1 and 2. Running

the maxSAT solvers for four hours resulted in finding solutions for two of the instances which could not be solved by the integer programming approach within one hour on a different environment. Although the integer programming method could also possibly find those solutions within four hours, the results show that maxSAT as an exact method gives promising results for employee scheduling problems. As many maxSAT solvers are publicly available and their performance is consistently improving, this approach could be useful to find solutions for employee scheduling problems.

5.3 Experimental Evaluation of the Hybrid Approach

Experiments for the evaluation of the hybrid approach were conducted within time limits of 10 and 60 minutes. Those limits were chosen so that the results could be easily compared with results produced by existing state of the art approaches from [CQ14]. Early benchmark experiments applied five repeated runs per instance within a 10 minute time limit and two repeated runs per instance within a 60 minute time limit. In the final experiments, ten repeated runs per instance were applied within a time limit of 10 minutes and five repeated runs per instance were applied within a time limit of 60 minutes.

In the following, section 5.3.1 describes how the algorithm's parameters were configured and tuned. Section 5.3.2 then presents the experimental results for different development stages of the proposed algorithm and draws a detailed comparison between results produced by the hybrid algorithm and results produced by state of the art approaches.

5.3.1 Algorithm Configuration

Several parameters take influence on the algorithm's performance. This section gives a short review on all the algorithm's parameters, which have already been described in chapter 4. Afterwards the configuration of those parameters is presented.

List of Parameters

Tabu list factor The tabu list factor takes an influence on the length of the used tabu list data structure that is used during local search. The longer the tabu list is, the more recent neighborhood moves will be stored and be marked as tabu whenever a new search move has to be selected. It can be advantageous if the length of a tabu list is dependent on the instance size of any given problem. Therefore, the tabu list factor is given to the algorithm as a real value between 0.0 and 1.0. The given value is then multiplied with numbers that are dependent on the instance size to determine an appropriate list length. The length of the used tabu list data structure is calculated as follows (where $|I|$ denotes the number of employees, h denotes the number of days in the schedule, $|T|$ denotes the number of shift types and $maximumBlockLength$ denotes the maximum length of consecutive shift assignments that is allowed by the problem's hard constraint):

$$tabuLength = |I| \cdot h \cdot |T| \cdot maximumBlockLength \cdot tabuListFactor \tag{5.1}$$

CP reset probability Whenever iterated local search enters the perturbation phase, some of the current solution's cells need to be emptied and marked for the CP method. The CP method will then try to find a good solution by using a forward checking search that considers all the marked cells. To select those cells, the algorithm will try to prefer those cells which are causing constraint violations. Some cells, however will be selected anyways depending on a random function. The cp reset probability parameter, which is given to the algorithm as a real value between 0.0 and 1.0, defines the probability of how likely a cell that is not involved with any constraint violation is selected for this purpose. Details about this selection process can be found in section 4.7.3.

α The evaluation of candidate solutions considers the number of hard constraint violations, the number of soft constraint influenced costs, and the number of cells which are included in hard constraint violations. The integer valued parameter α is used to control the impact of conflicted cells on the evaluation function (i.e. cells which are causing hard constraint violations). Details about the evaluation of candidate solutions are given in section 4.2.

β At the end of each search step, the embedded local search algorithm selects one of the moves out of the candidates that have been generated by the different neighborhoods. The selected neighborhood move will then be performed only under a certain acceptance probability. How likely it is that such a move candidate will be selected depends on multiple factors like the number of conflicted cells and changes in cost. The parameter β is an integer value which is multiplied by the cost changes that were induced by soft constraint violations. Details about the calculation of this acceptance probability is given in section 4.7.2.

Maximum iteration limit This limit defines how many consecutive iterations without cost improvements the embedded local search may perform before the perturbation procedure is called. This limit is given to the algorithm as an integer valued parameter.

Parameter Tuning

This section describes how the selection of the parameter values for the hybrid algorithm has been conducted.

Regarding the maximum iteration limit parameter, different values were used during early experiments to manually tune the parameter. Experiments with a value of 10000, 30000, 40000 and 80000 were performed on all instances under a time limit of 10 min. Since a value of 10000 gave the best results for most instances the maximum iteration limit parameter was set to 10000 for the remainder of the experiments. It could be a subject of future work to include systematical tuning also for this parameter.

The tabu list factor, cp reset probability, α, and β parameters have been tuned using the automated parameter tuning framework irace [LIDLSB11]. More specifically, irace version 1.07 has been used with the default settings. To tune the parameters, all 24 problem instances were given to irace. The maximum number of experiments was limited to 2000 and each run was limited to an execution time of 10 minutes. Parameter ranges given to the tuner were as follows: 0.0 to 1.0 for the tabu list factor and the CP reset probability parameter, 75 to 125 for the α parameter, and 800 to 1200 for the β parameter. Those ranges were based on some initial observations of manually started experiments.

The following elite configurations were determined by irace:

Table 5.6: Elite candidate parameter configurations determined by irace.

Tabu list factor	CP reset probability	α	β
0.3201	0.6991	98	1200
0.4610	0.6294	96	1179
0.3443	0.7370	98	1194
0.4997	0.7883	97	1155

Based on those results the following values were chosen for the final experiments: 0.5 for the tabu list factor, 0.7 for the CP reset probability, 100 for α and 1200 for β.

5.3.2 Experimental Results

This section presents the results produced by a number of experiments that were performed during and after the development of the hybrid algorithm. Additionally, at the end of this section a comparison between the gathered results and results from existing heuristic and exact solution techniques is drawn.

During the development of the hybrid algorithm for the employee scheduling problem additional components were added in multiple development stages. Several experiments were performed in between those development stages to track changes in the performance of the algorithm. The following subsections report on results that were obtained throughout the most important development stages.

Early Experiments using Simple Local Search

In early development phases there was no iterated local search procedure included in the algorithm. Basically the embedded local search that is described in 4.4 was solely used to solve the problem instances without any hybridization techniques and no construction heuristic. A simple random restart mechanism was included that restarted search with an empty solution after a given iteration limit. Experiments were conducted using a tabu list factor of 1.0 and an iteration limit of 80000. Table 5.7 shows the results of benchmark runs from this development stage using a time limit of 10 minutes.

Table 5.7: Experimental results from the first development stage. The columns show results that were produced by the algorithm in five independent runs, where each run had a time limit of 10 minutes. Results formatted in bold face denote proven optimal solutions.

	Run 1	Run 2	Run 3	Run 4	Run 5
Instance 1	**607**	**607**	**607**	**607**	**607**
Instance 2	926	929	925	928	922
Instance 3	1013	1018	1011	1013	1020
Instance 4	1722	1717	1716	1721	1721
Instance 5	1339	1341	1252	1252	1255
Instance 6	2459	3261	2658	2746	2556
Instance 7	1380	1395	1388	1375	1383
Instance 8	2339	2441	2332	2347	2333
Instance 9	719	613	691	699	595
Instance 10	5176	5254	5203	5278	5103
Instance 11	3794	4198	3885	3779	4411
Instance 12	6248	6552	6538	6677	8319
Instance 13	8978	9101	8966	8919	9579
Instance 14	-	-	-	-	-
Instance 15	12967	-	-	-	-
Instance 16	-	-	-	-	-
Instance 17	-	-	-	-	-
Instance 18	-	-	-	-	-
Instance 19	-	-	-	-	-
Instance 20	-	-	-	-	-
Instance 21	-	-	-	-	-
Instance 22	-	-	-	-	-
Instance 23	-	-	-	-	-
Instance 24	-	-	-	-	-

While those early experiments already produced decent solutions for the smaller instances, the algorithm could not produce feasible solutions for many of the larger instances.

Introducing Iterated Local Search with a CP based Perturbation

With the utilization of iterated local search techniques together with a perturbation procedure based on constraint programming, the algorithm could produce good results also for larger instances. In this stage a simple construction heuristic was implemented that generated an initial solution by simply filling the scheduling horizon greedily with shifts assignments based only on the cover requirements of the problem instance. Additionally, a basic variant of the acceptance mechanism that is described in section 4.7.2 was used that only considered cost changes produced by the violation of soft constraints.

To investigate the effects of the CP based perturbation procedure, at this development stage another variant of iterated local search was also considered, using a simple random perturbation strategy that just randomly reassigns conflicted cells.

Results which were produced using the CP based perturbation procedure are displayed in table 5.8 and 5.9. Results which were produced using the random perturbation procedure are displayed in tables 5.10 and 5.11. Table 5.12 compares the best results produced by both variants with results produced by a state of the art heuristic that is based on ejection chains. Additionally, the box plot in figure 5.1 shows a visualized comparison of the best outcomes for instances 1-13 scaled towards the best known solutions [Cur14].

The results from this development stage already provided promising results for many of the instances. The variant using a constraint programming based perturbation was able to determine good solutions and in many cases beats the heuristic based on ejection chains from the literature. However, at this stage the algorithm was not able to produce any feasible solution within a time limit of 60 minutes for instances 20-22.

Final Experiments

In the last phase of development the construction heuristic was extended to generate an initial solution that tries to satisfy as many hard constraint as possible. Additionally, the acceptance mechanism as well as the evaluation function were extended to consider changes in cost and the number of conflicted cell assignments. Details about the applied construction heuristic can be found in section 4.6. Information about the evaluation function and the acceptance mechanism can be found in sections 4.2 and 4.7.2.

The first set of experiments with the final implementation can be seen in tables 5.13 and 5.14. The following parameter values were used for those experiments: Tabu list factor = 1.0, CP reset probability = 0.1, $\alpha = 100$, $\beta = 1000$.

In the first set of the final experiments, the completed hybrid algorithm already was able to solve all of the instances and could improve many results that were produced by the state of the art heuristic based on ejection chains.

Table 5.8: Results produced using a CP based perturbation under a time limit of 10 minutes. The columns show results that were produced by the algorithm in five independent runs. Results formatted in bold face denote proven optimal solutions.

	Run 1	Run 2	Run 3	Run 4	Run 5
Instance 1	**607**	**607**	**607**	**607**	**607**
Instance 2	**828**	**828**	**828**	**828**	**828**
Instance 3	**1001**	1003	**1001**	1003	**1001**
Instance 4	1724	1734	1730	1722	1734
Instance 5	1238	1247	1238	1237	1244
Instance 6	2246	2245	2353	2357	2251
Instance 7	1187	1187	1187	1078	1078
Instance 8	1635	1929	1549	1668	-
Instance 9	461	461	468	455	462
Instance 10	4967	4769	4853	4769	4769
Instance 11	3567	3571	3459	3475	3473
Instance 12	4820	4856	4647	5024	4629
Instance 13	3461	3638	3921	3576	3527
Instance 14	1762	1668	1860	2085	-
Instance 15	5677	-	5470	4861	5522
Instance 16	3869	-	4151	4246	-
Instance 17	7237	-	7035	7121	7527
Instance 18	-	-	6618	5944	-
Instance 19	7491	-	7473	7665	6551
Instance 20	-	-	-	-	-
Instance 21	-	-	-	-	-
Instance 22	-	-	-	-	-
Instance 23	-	480064	485916	493543	498007
Instance 24	1208758	1264847	1208465	1202862	1212804

After the algorithm's parameters were tuned with irace, an extended set of experiments was conducted. For those experiments, ten repeated runs per instance were performed within a time limit of 10 minutes and five repeated runs per instance were performed within a time limit of 60 minutes.

The final results can be seen in tables 5.15 and 5.16. Table 5.17 shows a comparison of the best results acquired with state of the art results that use heuristic methods and exact approaches based on integer programming. A visual comparison of the best results from state of the art methods and the hybrid solver is shown in figure 5.2.

The final results show that the proposed hybrid solver is able to produce better results than the ejection chain based heuristic for all instances except for instance 24. Figure 5.2 shows a visualization of the best results from both algorithms scaled against the best

Table 5.9: Results produced using a CP based perturbation under a time limit of 60 minutes. The columns show results that were produced by the algorithm in two independent runs. Results formatted in bold face denote proven optimal solutions.

	Run 1	Run 2
Instance 1	**607**	**607**
Instance 2	**828**	**828**
Instance 3	**1001**	1002
Instance 4	1717	1718
Instance 5	1235	1236
Instance 6	2268	2165
Instance 7	1084	1072
Instance 8	1452	1446
Instance 9	455	455
Instance 10	4750	4851
Instance 11	3551	3462
Instance 12	4228	4216
Instance 13	3006	2767
Instance 14	1512	1615
Instance 15	4980	4737
Instance 16	-	3636
Instance 17	6710	6606
Instance 18	6118	5604
Instance 19	4573	5464
Instance 20	-	-
Instance 21	-	-
Instance 22	-	-
Instance 23	332659	321094
Instance 24	942501	955883

known upper bounds. An application of the wilcoxon signed-rank test over the scaled best results under the hypothesis that the median of the outcomes produced by the hybrid approach is less then the median of the outcomes produced by the ejection chain heuristic results in a p-value of almost 1. Therefore, it can be assumed that the proposed hybrid approach performs significantly better than the state of the art heuristic under a very high probability.

Furthermore, the hybrid approach delivers competitive results also when compared with results produced by state of the art exact methods. The obtained results are comparable with those produced by Gurobi for all of the considered instances and the hybrid approach could further improve results for eight of the instances.

Additionally, new upper bounds could be acquired for instances 19-24 by giving the

Table 5.10: Results produced using a random perturbation procedure under a time limit of 10 minutes. The columns show results that were produced by the algorithm in five independent runs. Results formatted in bold face denote proven optimal solutions.

	Run 1	Run 2	Run 3	Run 4	Run 5
Instance 1	**607**	**607**	**607**	**607**	**607**
Instance 2	829	**828**	**828**	**828**	**828**
Instance 3	**1001**	1003	**1001**	1003	1003
Instance 4	1723	1721	1724	1723	1722
Instance 5	1247	1245	1342	1244	1247
Instance 6	2254	2345	2453	2257	2351
Instance 7	1176	1176	1280	1278	1278
Instance 8	-	-	-	-	-
Instance 9	555	560	466	469	470
Instance 10	-	5076	4973	5167	4960
Instance 11	3578	3580	3672	3580	3672
Instance 12	4736	-	4736	4538	-
Instance 13	3639	3750	3647	3921	3568
Instance 14	-	-	-	-	-
Instance 15	-	-	-	-	-
Instance 16	4057	4057	-	-	4151
Instance 17	-	-	7121	7237	6902
Instance 18	-	-	5525	-	-
Instance 19	6654	-	-	-	-
Instance 20	-	-	-	-	-
Instance 21	-	-	-	-	-
Instance 22	-	-	-	-	-
Instance 23	507211	-	494133	488156	488396
Instance 24	1264847	1211463	1264847	1208465	1215276

hybrid approach an extended time limit of three days. The current best bounds that are known for all instances are available at [Cur14].

Table 5.11: Results produced using a random perturbation procedure under a time limit of 60 minutes. The columns show results that were produced by the algorithm in two independent runs. Results formatted in bold face denote proven optimal solutions.

	Run 1	Run 2
Instance 1	**607**	**607**
Instance 2	**828**	**828**
Instance 3	1003	1003
Instance 4	1718	1719
Instance 5	1237	1240
Instance 6	2258	2159
Instance 7	1178	1180
Instance 8	1886	-
Instance 9	479	475
Instance 10	4997	4875
Instance 11	3494	-
Instance 12	4768	-
Instance 13	3066	2801
Instance 14	-	-
Instance 15	-	-
Instance 16	-	-
Instance 17	6916	-
Instance 18	-	5509
Instance 19	-	4748
Instance 20	-	-
Instance 21	82541	-
Instance 22	-	-
Instance 23	320788	332659
Instance 24	940803	-

Table 5.12: This table compares the best results produced by a CP based perturbation with the best results that were produced using a random perturbation technique. Additionally, results produced by a metaheuristic method from the literature [CQ14] based on ejection chains are displayed. Results formatted in bold face denote proven optimal solutions.

Instance	CP perturbation		Random perturbation		Ejection Chain [CQ14]	
	10 min	60 min	10 min	60 min	10 min	60 min
Instance 1	**607**	**607**	**607**	**607**	**607**	**607**
Instance 2	**828**	**828**	**828**	**828**	923	837
Instance 3	**1001**	**1001**	**1001**	1003	1003	1003
Instance 4	1722	1717	1721	1718	1719	1718
Instance 5	1237	1235	1244	1237	1439	1358
Instance 6	2245	2165	2254	2159	2344	2258
Instance 7	1078	1072	1176	1178	1284	1269
Instance 8	1549	1446	-	1886	2529	2260
Instance 9	455	455	466	475	474	463
Instance 10	4769	4750	4960	4875	4999	4797
Instance 11	3459	3462	3578	3494	3967	3661
Instance 12	4629	4216	4538	4768	5611	5211
Instance 13	3461	2767	3568	2801	8707	3037
Instance 14	1668	1512	-	-	2542	1847
Instance 15	4861	4737	-	-	6049	5935
Instance 16	3869	3636	4057	-	4343	4048
Instance 17	7035	6606	6902	6916	7835	7835
Instance 18	5944	5604	5525	5509	6404	6404
Instance 19	6551	4573	6654	4748	6522	5531
Instance 20	-	-	-	-	23531	9750
Instance 21	-	-	-	82541	38294	36688
Instance 22	-	-	-	-	-	516686
Instance 23	480064	321094	488156	320788	-	54384
Instance 24	1202862	942501	1208465	940803	-	156858

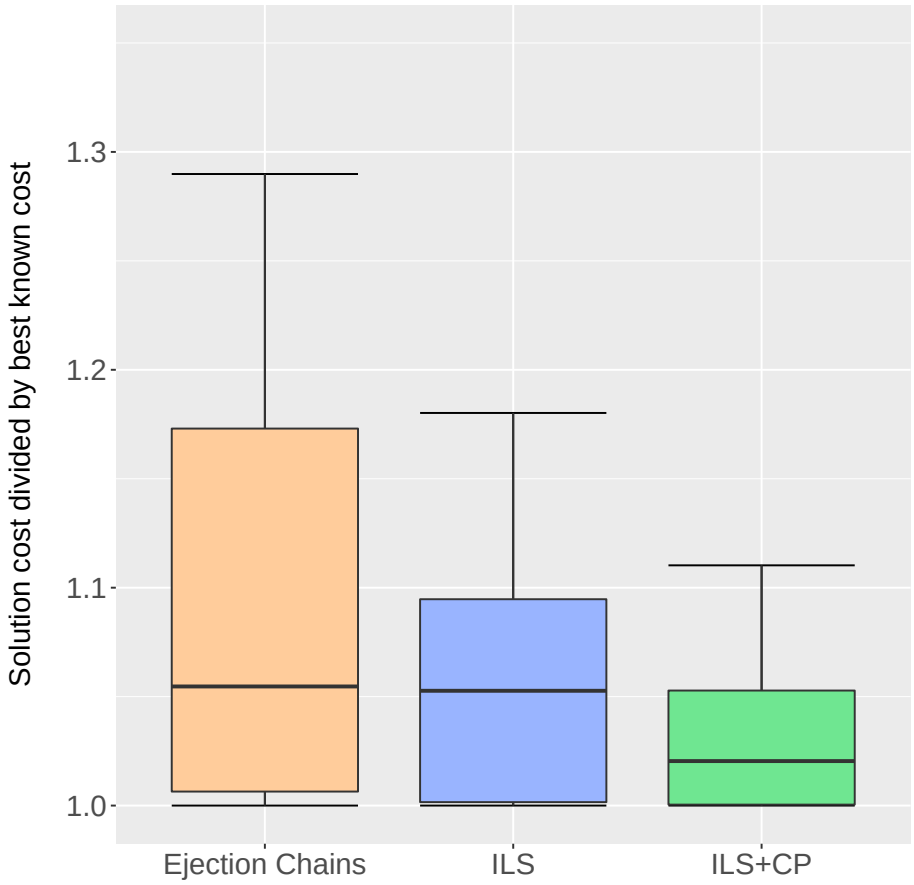

Figure 5.1: This figure compares the best results produced by iterated local search using a CP based perturbation (ILS+CP) and a random perturbation technique (ILS) with the best results from the ejection chain based metaheuristic (Ejection Chains). Only results for instances 1-13 have been taken into account because the result set regarding larger instances was incomplete for some of the algorithms. Results were scaled against the all time best known solutions (available at [Cur14]), where a value of 1.0 means that the best known solution could be reached.

Table 5.13: First set of results produced using the complete hybrid algorithm within a time limit of 10 minutes. The columns show results that were produced by the algorithm in five independent runs. Results formatted in bold face denote proven optimal solutions.

	Run 1	Run 2	Run 3	Run 4	Run 5
Instance 1	**607**	**607**	**607**	**607**	**607**
Instance 2	829	**828**	**828**	**828**	**828**
Instance 3	1003	1003	1003	1003	1004
Instance 4	1815	1720	1722	1719	1722
Instance 5	1237	1240	1241	1245	1243
Instance 6	2250	2155	2266	2250	2163
Instance 7	1080	1087	1189	1283	1086
Instance 8	1658	1844	1663	1652	1863
Instance 9	460	462	466	473	459
Instance 10	4887	4864	4884	4865	4860
Instance 11	3483	3477	3474	3467	3482
Instance 12	5346	5148	5153	5153	5280
Instance 13	3548	4089	3543	3536	3659
Instance 14	1544	1433	1741	1544	1939
Instance 15	5688	5463	5860	5368	5671
Instance 16	4258	3850	4071	3664	4258
Instance 17	7345	7325	8147	7325	7313
Instance 18	5921	5611	5921	5611	6643
Instance 19	5838	5068	5147	5509	5838
Instance 20	9957	-	10132	10847	10006
Instance 21	-	-	-	43355	-
Instance 22	-	-	211956	-	-
Instance 23	267836	269953	260760	260703	260341
Instance 24	844619	783109	-	792434	877646

Table 5.14: First set of results produced using the complete hybrid algorithm within a time limit of 60 minutes. The columns show results that were produced by the algorithm in two independent runs. Results formatted in bold face denote proven optimal solutions.

	Run 1	Run 2
Instance 1	**607**	**607**
Instance 2	**828**	**828**
Instance 3	1003	**1001**
Instance 4	1718	1718
Instance 5	1239	1239
Instance 6	2066	2149
Instance 7	1106	1181
Instance 8	1550	1534
Instance 9	459	456
Instance 10	4680	4778
Instance 11	3473	3486
Instance 12	4455	5164
Instance 13	3530	3424
Instance 14	1530	1541
Instance 15	4890	5069
Instance 16	3759	3749
Instance 17	6506	6605
Instance 18	5615	5600
Instance 19	4922	5417
Instance 20	7006	7569
Instance 21	23571	23622
Instance 22	54087	63284
Instance 23	53322	55081
Instance 24	246570	251360

Table 5.15: Experimental results using the hybrid algorithm's final version within a time limit of 10 minutes. The columns show results that were produced by the algorithm in ten independent runs. Results formatted in bold face denote proven optimal solutions.

	Run 1	Run 2	Run 3	Run 4	Run 5	Run 6	Run 7	Run 8	Run 9	Run 10
Inst. 1	**607**	**607**	**607**	**607**	**607**	**607**	**607**	**607**	**607**	**607**
Inst. 2	**828**	**828**	**828**	**828**	**828**	**828**	**828**	**828**	**828**	**828**
Inst. 3	1003	1003	**1001**	1003	1003	1003	1003	1002	1002	1002
Inst. 4	1720	1719	1719	1721	1722	1721	**1716**	1722	1723	1719
Inst. 5	1249	1245	1242	1243	1246	1244	1237	1237	1150	1246
Inst. 6	2247	2156	2251	2249	2164	2250	2145	2161	2161	2154
Inst. 7	1182	1095	1171	1177	1091	1191	1090	1183	1172	1176
Inst. 8	1644	1548	1661	1668	1655	1554	1567	1647	1644	1662
Inst. 9	454	459	465	462	456	455	464	461	461	459
Inst. 10	4871	4866	4660	4955	4859	4761	4769	4878	4863	4878
Inst. 11	3478	3472	3479	3470	3488	3475	3476	3476	3478	3570
Inst. 12	4519	4429	4833	4338	4746	4636	4535	4430	4733	4535
Inst. 13	3468	3671	3157	3365	3552	3577	3631	3755	3458	3652
Inst. 14	1632	1626	1731	1623	1450	1743	1728	1637	1733	1430
Inst. 15	5052	5280	4967	5467	4871	4943	4943	5056	4971	4962
Inst. 16	4062	4258	4253	4151	3834	3834	4165	4366	4248	3754
Inst. 17	7214	7335	7017	7014	7621	7020	6724	7010	7520	6720
Inst. 18	5914	5995	5813	5400	6005	6112	5920	5519	6113	6717
Inst. 19	5192	5555	5185	5085	4780	5166	4861	5269	5171	5166
Inst. 20	11591	9162	9583	10575	9365	8763	12253	10575	9591	9595
Inst. 21	33163	-	168833	-	48304	52630	51383	-	53872	42602
Inst. 22	-	-	-	-	-	192946	-	-	-	-
Inst. 23	199376	195233	198980	189850	195744	196694	199194	201523	202131	227209
Inst. 24	534093	525837	526984	519173	522333	522182	523493	520832	525675	527633

Table 5.16: Experimental results using the hybrid algorithm's final version within a time limit of 60 minutes. The columns show results that were produced by the algorithm in five independent runs. Results formatted in bold face denote proven optimal solutions.

	Run 1	Run 2	Run 3	Run 4	Run 5
Instance 1	**607**	**607**	**607**	**607**	**607**
Instance 2	**828**	**828**	**828**	**828**	**828**
Instance 3	1002	**1001**	**1001**	**1001**	1002
Instance 4	1718	**1716**	1717	1717	1717
Instance 5	1156	1236	1147	1241	1236
Instance 6	2150	2050	2148	2070	2148
Instance 7	1093	1087	1100	1086	1084
Instance 8	1566	1554	1470	1464	1552
Instance 9	456	456	454	461	454
Instance 10	4763	4667	4766	4676	4761
Instance 11	3465	3467	3473	3460	3457
Instance 12	4348	4308	4332	4549	4338
Instance 13	3149	2961	3319	3269	3249
Instance 14	1529	1535	1432	1530	1528
Instance 15	4851	4570	5056	4770	5064
Instance 16	3847	3756	3748	3853	3949
Instance 17	6909	6609	6800	6894	6900
Instance 18	5523	5787	5696	5611	5416
Instance 19	4544	4639	4543	4693	4364
Instance 20	7216	6654	6895	6992	7078
Instance 21	-	25599	23162	28265	22549
Instance 22	-	56587	49218	48382	69274
Instance 23	41158	38337	39011	38957	38614
Instance 24	177037	177833	180891	194325	191083

Table 5.17: This table compares the best results produced with the proposed hybrid approach with results produced by methods from [CQ14]. Results formatted in bold face denote proven optimal solutions.

Instance	Hybrid Solver		Ejection Chain		Branch & Price	Gurobi
	10 min	60 min	10 min	60 min		
Instance1	**607**	**607**	**607**	**607**	**607**	**607**
Instance2	**828**	**828**	923	837	**828**	**828**
Instance3	**1001**	**1001**	1003	1003	**1001**	**1001**
Instance4	**1716**	**1716**	1719	1718	**1716**	**1716**
Instance5	1150	1147	1439	1358	1160	**1143**
Instance6	2145	2050	2344	2258	1952	**1950**
Instance7	1090	1084	1284	1269	1058	**1056**
Instance8	1548	1464	2529	2260	1308	1323
Instance9	454	454	474	463	439	439
Instance10	4660	4667	4999	4797	**4631**	**4631**
Instance11	3470	3457	3967	3661	**3443**	**3443**
Instance12	4338	4308	5611	5211	4046	**4040**
Instance13	3157	2961	8707	3037	-	3109
Instance14	1430	1432	2542	1847	-	1280
Instance15	4871	4570	6049	5935	-	4964
Instance16	3754	3748	4343	4048	3323	3233
Instance17	6720	6609	7835	7835	-	5851
Instance18	5400	5416	6404	6404	-	4760
Instance19	4780	4364	6522	5531	-	5420
Instance20	8763	6654	23531	9750	-	-
Instance21	33163	22549	38294	36688	-	-
Instance22	192946	48382	-	516686	-	-
Instance23	189850	38337	-	54384	-	-
Instance24	519173	177037	-	156858	-	-

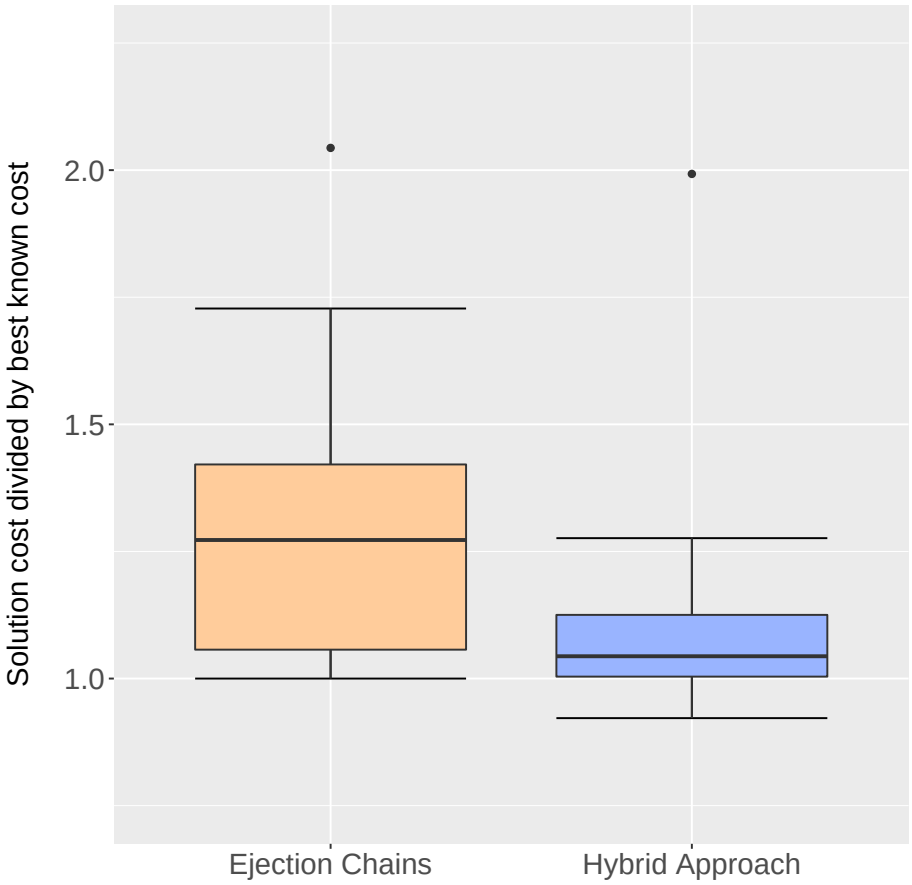

Figure 5.2: This figure compares the best results that were produced by the proposed hybrid approach with the best results produced by a state of the art heuristic method which is based on ejection chains [CQ14]. Results were scaled against the all time best known solutions (available at [Cur14]), where a value of 1.0 means that the best known solution could be reached. (One massive outlier for the ejection chain based heuristic at point 13.63 has been removed for formatting reasons.)

CHAPTER 6

Conclusion

In this thesis two novel solution methods have been proposed to solve a well known variant of the employee scheduling problem.

The first approach introduced for the first time a partial weighted Boolean maximum satisfiability model for an employee scheduling problem. With the proposed model, a number of maxSAT instances were generated using four different cardinality constraint encoding methods. Additionally, the effects of the different cardinality encoding methods have been compared on two leading maxSAT solvers. It has been shown that there is a need to experimentally select an efficient combination of cardinality constraint encodings for each solver separately. A comparison between the two solvers could not point out a clear winner for all of the considered benchmark tests. While WPM3 performed better on smaller instances, Optiriss was able to produce better results for many of the larger instances.

Currently an exact approach based on integer programming provides better results than maxSAT for most of the considered instances. However, maxSAT could provide optimal solutions for two of the instances and obtained solutions for two very large instances within four hours, which could not be solved by integer programming within one hour. Therefore, as nowadays different maxSAT solvers are available and their performance is consistently improving, exact maxSAT techniques can be useful for solving employee scheduling problems in the future.

The second approach that has been proposed in this thesis introduced a novel method of using iterated local search together with a constraint programming based perturbation function. Within this method, appropriate search neighborhoods from the literature have been combined and utilized using a min conflicts based local search as well as a novel construction heuristic. Furthermore, we introduce a novel constraint programming approach for solving sub-problems of the considered employee scheduling problem. The approach has delivered very good results for the considered employee scheduling problem.

Parameters have been tuned using an automatic parameter tuning software and an empirical evaluation has shown that the results can compete with both heuristic as well exact methods that have been proposed in the recent literature. Further, results produced by state of the art techniques were improved for 6 out of 24 problem instances. With the use of the hybrid approach five new unknown upper bounds for the instances from [Cur14] have been provided.

Possible improvements and extensions concerning the proposed maxSAT model could be subject of future work. For example, it would be interesting to investigate if symmetries can be broken. Apart from that, using the results of the simplified instances as a starting point for local search could be beneficial and a hybridization of maxSAT with heuristic techniques within the framework of very large neighborhood search could be considered. Regarding the hybrid approach it would be interesting to investigate the performance of other techniques that could be used within iterated local search instead of constraint programming. For example, one could also consider the use of maxSAT as a perturbation method.

List of Figures

List of Tables

List of Algorithms

Glossary

Glossary

branch and bound Branch and bound is an algorithm design paradigm that uses a systematic enumeration of candidate solutions. Branches of the search tree are only enumerated if estimated lower and upper bounds can be produced by the candidate solutions of the branch [RvBW06]. 33, 34

conjunctive normal form A Boolean logic formula is in conjunctive normal form if it is a conjunction of clauses, where a clause is a disjunction of literals [BHvMW09]. 12

constraint programming Constraint programming is a programming paradigm where relations between variables are stated in form of constraints [RvBW06]. 3, 8, 11, 12, 23, 33–35, 37–39, 52, 65, 66

constraint propagation Constraint propagation methods try to enforce local consistency conditions in constraint satisfaction problems. Local consistency conditions are properties related to the consistency of subsets of variables and constraints [RvBW06]. 11, 33, 34

construction heuristic A construction heuristic starts with an empty solution and repeatedly extends the current solution using heuristics until a complete solution is obtained [GK06]. 23, 36–38, 50, 52, 65

decision variables The main variables of a problem are called decision variables. The aim of the problem is to decide which values should be assigned those variables [BHvMW09]. 9, 15, 18

dynamic programming Dynamic programming is a method for solving a complex problem by breaking it down into a collection of simpler subproblems and solving each of those subproblems just once [RvBW06]. 11

ejection chains Ejection chain methods describe heuristic search techniques, that chain together sequences of paired steps as their search moves. The first component of

each paired step always creates an inducement for further change, while the second component tries to restore the solution [Glo96]. 2, 9, 11, 52, 57

forward checking Forward checking is a look ahead mechanism for backtracking search. After each variable assignment, it checks whether other variables can take values that are consistent with the assignment [RvBW06]. 12, 34, 49

hill climbing Hill climbing is an iterative algorithm that starts with an arbitrary solution to an optimization problem and attempts to find a better solution by incrementally changing a single element of the solution [HS04]. 11

integer programming Integer programming deals with solution strategies for mathematical optimization problems in which some or all of the variables are restricted to be integers [RvBW06]. 2, 9, 41, 42, 47, 48, 53, 65

iterated local search Iterated local search defines a modification of local search. An iterative sequence of perturbing a solution and then performing local search enhances the possibilities of the search to escape local optimal solutions [LMS01]. 3, 23, 33, 37, 39, 49, 50, 52, 65, 66

local search Local search algorithms move from solution to solution in the search space by applying local changes, until a supposedly optimal solution is found or a time limit has passed [HS04]. 12, 23–25, 27, 28, 30, 31, 33, 35, 37, 38, 48–50, 65, 66

mathematical programming Mathematical programming means to use mathematical optimization models in order to assist in taking problem related decisions [RvBW06]. 1, 8, 9

maximum satisfiability problem The maximum satisfiability problem is the problem of determining the maximum number of clauses of a given Boolean formula that can be made true by an assignment of truth values [BHvMW09]. 1

metaheuristic A metaheuristic describes a higher-level heuristic designed to obtain good solutions for optimization problems. Such a heuristic makes only few assumptions about the concrete problem being solved and can therefore be used for a variety of different problems [GK06]. 1, 2, 8, 9, 57, 58, 67

min conflicts heuristic The min conflicts heuristic is a heuristic repair algorithm that tries to improve candidate solutions by performing neighborhood moves that reduce the number of conflicts in the given solution [MJPL92]. 28, 30, 31, 37, 65

NP-hardness NP-hard problems denote a class of problems in complexity theory that are considered to be at least as hard as any problem in the class NP. NP includes problems that are solvable in non deterministic polynomial time [PGW$^+$96]. 1, 12

satisfiability problem The Boolean satisfiability problem is the problem of determining if there exists a truth value assignment that satisfies a given formula [BHvMW09]. 1

simulated annealing Simulated annealing is a probabilistic metaheuristic to approximate the global optimum of a function [GK06]. 12, 38

tabu search Tabu search is a metaheuristic method, that tries to escape local optima by preventing the repetition of recently or frequently performed search moves in local search [GK06]. 12

variable depth search A variant of local search, where multiple neighborhood moves are allowed within one search step [Glo96]. 11

Acronyms

Acronyms

CNF conjuctive normal form. 12, *Glossary:* conjunctive normal form

CP constraint programming. 11, 31, 33, 49, 52, *Glossary:* constraint programming

DIMACS center for discrete mathematics & theoretical computer science. 43

ILS iterated local search. *Glossary:* iterated local search

IP integer programming. 2, 9, 11, *Glossary:* integer programming

maxSAT maximum satisfiability problem. 1–3, 8, 12, 13, 15–17, 19–21, 41, 43, 44, 47, 48, 65, 66, *Glossary:* maximum satisfiability problem

SAT satisfiability problem. 1, 8, 12, 13, *Glossary:* satisfiability problem

VDS variable depth search. 11, *Glossary:* variable depth search

Bibliography

[ADG15] Carlos Ansótegui, Frédéric Didier, and Joel Gabàs. Exploiting the structure of unsatisfiable cores in maxsat. In *Proceedings of the Twenty-Fourth International Joint Conference on Artificial Intelligence, IJCAI 2015, Buenos Aires, Argentina, July 25-31, 2015*, pages 283–289, 2015.

[aH00] Harald Meyer auf'm Hofe. Solving rostering tasks as constraint optimization. In *Practice and Theory of Automated Timetabling III, Third International Conference, PATAT 2000, Konstanz, Germany, August 16-18, 2000, Selected Papers*, pages 191–212, 2000.

[ANOR09] Roberto Asín, Robert Nieuwenhuis, Albert Oliveras, and Enric Rodríguez-Carbonell. Cardinality networks and their applications. In *Theory and Applications of Satisfiability Testing - SAT 2009, 12th International Conference, SAT 2009, Swansea, UK, June 30 - July 3, 2009. Proceedings*, pages 167–180, 2009.

[BC14] Edmund K. Burke and Timothy Curtois. New approaches to nurse rostering benchmark instances. *European Journal of Operational Research*, 237(1):71–81, 2014.

[BCP+08] Edmund K. Burke, Timothy Curtois, Gerhard F. Post, Rong Qu, and Bart Veltman. A hybrid heuristic ordering and variable neighbourhood search for the nurse rostering problem. *European Journal of Operational Research*, (2):330–341, 2008.

[BCQB10] Edmund K. Burke, Timothy Curtois, Rong Qu, and Greet Vanden Berghe. A scatter search methodology for the nurse rostering problem. *JORS*, 61(11):1667–1679, 2010.

[BCQB13] Edmund K. Burke, Timothy Curtois, Rong Qu, and Greet Vanden Berghe. A time predefined variable depth search for nurse rostering. *INFORMS Journal on Computing*, 25(3):411–419, 2013.

[BGM+10] Andreas Beer, Johannes Gärtner, Nysret Musliu, Werner Schafhauser, and Wolfgang Slany. An ai-based break-scheduling system for supervisory personnel. *IEEE Intelligent Systems*, 25(2):60–73, 2010.

[BGSV15] Miquel Bofill, Marc Garcia, Josep Suy, and Mateu Villaret. Maxsat-based scheduling of B2B meetings. In *Integration of AI and OR Techniques in Constraint Programming - 12th International Conference, CPAIOR 2015, Barcelona, Spain, May 18-22, 2015, Proceedings*, pages 65–73, 2015.

[BHvMW09] Armin Biere, Marijn Heule, Hans van Maaren, and Toby Walsh, editors. *Handbook of Satisfiability*, Frontiers in Artificial Intelligence and Applications. IOS Press, 2009.

[BLQ10] Edmund K. Burke, Jingpeng Li, and Rong Qu. A hybrid model of integer programming and variable neighbourhood search for highly-constrained nurse rostering problems. *European Journal of Operational Research*, 203(2):484–493, 2010.

[CQ14] Tim Curtois and Rong Qu. Computational results on new staff scheduling benchmark instances. Technical report, ASAP Research Group, School of Computer Science, University of Nottingham, NG8 1BB, Nottingham, UK, October 2014.

[Cur14] Timothy Curtois. Staff scheduling benchmark instances, 2014. `http://www.cs.nott.ac.uk/~psztc/NRP/index.html`, Accessed: 2016-03-07.

[dBBB+13] Jorne Van den Bergh, Jeroen Beliën, Philippe De Bruecker, Erik Demeulemeester, and Liesje De Boeck. Personnel scheduling: A literature review. *European Journal of Operational Research*, (3):367–385, 2013.

[DM14] Emir Demirovic and Nysret Musliu. Modeling high school timetabling as partial weighted maxsat. In *LaSh 2014: The 4th Workshop on Logic and Search (a SAT / ICLP workshop at FLoC 2014), July 18, Vienna, Austria*, 2014.

[EJKS04] Andreas T. Ernst, Houyuan Jiang, Mohan Krishnamoorthy, and David Sier. Staff scheduling and rostering: A review of applications, methods and models. *European Journal of Operational Research*, (1):3–27, 2004.

[GK06] Fred W. Glover and Gary A. Kochenberger. *Handbook of metaheuristics*, volume 57. Springer Science & Business Media, 2006.

[Glo96] Fred Glover. Ejection chains, reference structures and alternating path methods for traveling salesman problems. *Discrete Applied Mathematics*, 65(1):223–253, 1996.

[GO15] Inc. Gurobi Optimization. Gurobi optimizer reference manual, 2015.

[HS04] Holger H Hoos and Thomas Stützle. *Stochastic local search: Foundations & applications*. Elsevier, 2004.

[Kir84] Scott Kirkpatrick. Optimization by simulated annealing: Quantitative studies. *Journal of Statistical Physics*, 34(5-6):975–986, 1984.

[KKMS15] Lucas Kahlert, Franziska Krüger, Norbert Manthey, and Aaron Stephan. Riss solver framework v5. 05. *SAT-Race*, 2015.

[LD10] Ailsa H. Land and Alison G. Doig. An automatic method for solving discrete programming problems. In *50 Years of Integer Programming 1958-2008 - From the Early Years to the State-of-the-Art*, pages 105–132. 2010.

[LIDLSB11] Manuel López-Ibáñez, Jérémie Dubois-Lacoste, Thomas Stützle, and Mauro Birattari. The irace package, iterated race for automatic algorithm configuration. Technical Report TR/IRIDIA/2011-004, IRIDIA, Université Libre de Bruxelles, Belgium, 2011.

[LMS01] Helena R Lourenço, Olivier Martin, and Thomas Stützle. A beginner's introduction to iterated local search. In *Proceedings of MIC*, volume 2, pages 1–6, Porto, Portugal, July 2001.

[max15] Max-sat evaluation 2015, 2015. `http://www.maxsat.udl.cat/15/`, Accessed: 2016-22-07.

[MJPL92] Steven Minton, Mark D. Johnston, Andrew B. Philips, and Philip Laird. Minimizing conflicts: A heuristic repair method for constraint satisfaction and scheduling problems. *Artif. Intell.*, 58(1-3):161–205, 1992.

[MML14] Ruben Martins, Vasco M. Manquinho, and Inês Lynce. Open-wbo: A modular maxsat solver,. In *Theory and Applications of Satisfiability Testing - SAT 2014 - 17th International Conference, Held as Part of the Vienna Summer of Logic, VSL 2014, Vienna, Austria, July 14-17, 2014. Proceedings*, pages 438–445, 2014.

[Mus06] Nysret Musliu. Heuristic methods for automatic rotating workforce scheduling. *International Journal of Computational Intelligence Research*, 2(4):309–326, 2006.

[PGFP09] Javier Puente, Alberto Gomez, Isabel Fernández, and Paolo Priore. Medical doctor rostering problem in a hospital emergency department by means of genetic algorithms. *Computers & Industrial Engineering*, 56(4):1232–1242, 2009.

[PGW+96] Christos H. Papadimitriou, Oded Goldreich, Avi Wigderson, Alexander A. Razborov, and Michael Sipser. The future of computational complexity theory: part I. *SIGACT News*, 27(3):6–12, 1996.

[RvBW06] Francesca Rossi, Peter van Beek, and Toby Walsh, editors. *Handbook of Constraint Programming*, volume 2 of *Foundations of Artificial Intelligence*. Elsevier, 2006.

[Sin05] Carsten Sinz. Towards an optimal CNF encoding of boolean cardinality constraints. In *Principles and Practice of Constraint Programming - CP 2005, 11th International Conference, CP 2005, Sitges, Spain, October 1-5, 2005, Proceedings*, pages 827–831, 2005.

[Stü97] Thomas Stützle. Lokale suchverfahren für constrain satisfaction probleme: die *min conflicts* heuristik und tabu search. *KI*, 11(1):14–20, 1997.