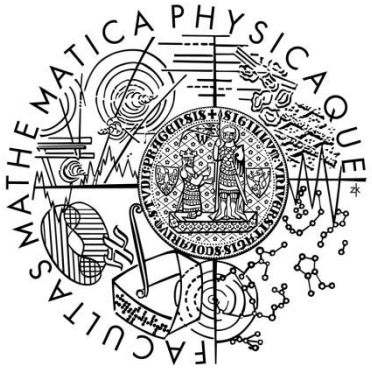
Charles University in Prague

Faculty of Mathematics and Physics

MASTER THESIS



Štefan Pacinda

Workforce Optimisation

Department of Theoretical Computer Science and Mathematical Logic (KTIML)

doc. RNDr. Roman Barták Ph.D.

Study Programme: Informatics

Specialization: Software Systems

Prague 2011

Acknowledgement:

I wish to thank RNDr. Karel Jelinek, Ph.D., for his detailed review, constructive criticism and excellent advice during the preparation of this thesis.

I would also like to thank my supervisor, doc. RNDr. Roman Barták, Ph.D., for his essential comments, and for his important support throughout this work.

Prague, Czech Republic, April 2011

Stefan Pacinda

I declare that I carried out this master thesis independently, and only with the cited sources, literature and other professional sources.

I understand that my work relates to the rights and obligations under the Act No. 121/2000 Coll., the Copyright Act, as amended, in particular the fact that the Charles University in Prague has the right to conclude a license agreement on the use of this work as a school work pursuant to Section 60 paragraph 1 of the Copyright Act.

In Prague April 15, 2011 Štefan Pacinda

Název práce: Optimalizace pracovních sil

Autor: Štefan Pacinda

Katedra / Ústav: Katedra teoretické informatiky a matematické logiky

Vedoucí diplomové práce: doc. RNDr. Roman Barták Ph.D., KTIML

Abstrakt: Optimalizace pracovních sil se zabývá udržováním produktivity pracovní síly například v nemocnicích, či dopravních společnostech. Zahrnuje problém rozhodování o nutnosti použití jednotlivých dovedností v každém okamžiku, stejně jako rozhodování o množství vyžadovaného personálu. Po těchto rozhodnutí následuje řešení problému alokování určitých zaměstnanců ke směnám za současného plnění požadavků na dovednosti ale i jiných omezení odvozených z na příklad právních regulací, odborových dohod a vlastních preferencí. Tato práce se zabývá optimalizací pracovních sil, tedy optimálním nasazením personálu na směny za účelem pokrytí požadavků na zdroje, které se mění v čase. V této práci je řešený problém detailně rozebrán, popsán a namodelován jako problém celočíselného programování. Jsou zde prezentovány detaily o implementaci a vyčerpávající analýza a experimenty s instancí problém z praxe za účelem prokázání naplnění cílů, které si práce klade.

Klíčová slova: Rostering, Optimalizace pracovních sil, Rozvrhování směn

Title: Workforce Optimisation

Author: Štefan Pacinda

Department / Institute:
Department of Theoretical Computer Science and Mathematical Logic (KTIML)

Supervisor of the master thesis: doc. RNDr. Roman Barták Ph.D., KTIML

Abstract: Workforce management deals with the problem of maintaining productive workforce for example in call centers, hospitals, transportation companies etc. It includes the problem of deciding which skills are necessarily at each given time and how many personnel with given skills is required. These decisions are followed by solving the problem of allocating particular employees to shifts while satisfying the skill demands but also other constraints derived for example from law regulations, trade unions agreements, and individual preferences. This thesis deals with workforce optimization, that is with the optimal assignment of personnel to shifts in order to cover the demand for resources that vary over time. In this paper the solved problem is described in all detail and modeled as mixed integer program. Implementation details are presented and exhaustive analysis and experiments on a real life problem instance are performed to assure that the aims of the work have been met.

Keywords: Rostering, Workforce Management, Shift Scheduling

Contents

Introduction 5

Workforce Management 5

1.1. Contribution 5

1.2. Literature 6

1.3. Outline 7

Chapter 2. Formal Description of the Solved Problem 8

2.1. Entities, Attributes and Relations 8

2.2. Roster Features 19

2.3. Objectives Overview 29

Chapter 3. Mixed Integer Model 30

3.1. Mixed Integer Program 30

3.2. Constraints Classification 31

3.3. Variables 33

3.4. Objective Function 36

3.5. Implemented Constraints 38

3.6. Complete Model Recap 60

Chapter 4. Implementation 65

4.1. Architecture 65

4.2. Testing 66

Chapter 5. Justification 68

5.1. Problem Instance 68

5.2. Computation 76

5.3. Results 77

Chapter 6. Conclusion 83

Bibliography..... 84

Introduction

Workforce Management

Workforce management (WFM) encompasses various activities to achieve productivity of workforce. In this thesis we will focus on one of these activities – shift scheduling. Shift scheduling problem constitutes of assigning employees to working shifts on various activities based on resource requirements subject to various conditions and restrictions. On the top of satisfying all requirements and conditions we want schedules to be possibly optimal given a set of goals to optimize against. The process of generating these rosters has to be reasonably efficient.

This thesis is based on a real world project, where the customer is a company with many departments including an in-house contact center. Within the scope of this thesis we will be designing an application that will be able to generate rosters for employees working in back-office of this contact center. Later the very same product will be used to schedule staff for different sites of the enterprise with no needs to alter the application.

1.1. Contribution

In this paper we deal with scheduling workforce which is one of the classical disciplines of operational research. The topic most often examined is scheduling hospital staff. There is relatively large number of papers on nurse and physician scheduling. The motivation for examining workforce scheduling is in most cases economic. Authors usually try to find optimal rosters in terms of staffing costs with respect to some necessary staffing level restrictions.

A common property of hospital staff scheduling is the lack of multi-skill environment. Nurses are usually scheduled within the scope of one ward and the skills of nurses are not taken into account as they are more or less the same. The first contribution of this work is the fact that we allow the arbitrary number of activities to be introduced into the system which increases the size of problem substantially. We also support quite complex set of preferences, requirement types, priorities and other roster features that are not usually taken into account when scheduling workforce.

Our second contribution is that we pursue rather different optimization criteria than we usually see in literature. The staffing levels are relatively stable and the staffing costs are of no interest to us. The aim of the optimization is the quality of rosters

from both employees' and enterprise's point of view. Employees are allowed to state various requests and preferences on shift composition, time off distribution etc. Similarly, the managers can set some preferences and priorities and limits on scheduling employees to some types of shifts as well as determine the optimal composition of shifts. The joint fulfillment of all these wishes and needs is the objective of the optimization in this work. However, it doesn't foreclose the possibility of adding staffing cost to the optimizing criteria, if requested.

The following contribution is strictly practical. The software that is going to be the ultimate output of this work is going to automate the roster creation process that has been done manually so far. Both the data collection and the roster generation will be automated. As a result, the errors caused by human factor combined with high complexity of some roster features will be eradicated too.

Last but not least we want to mention the possibility of parameterization of the computation. It is possible to configure the same deployed version of the software to suit different customers with rather different expectations about importance of various features and conception of trade-off among various criteria. This configuration is done by setting parameters on the fly.

1.2. Literature

The literature offers various approaches and algorithms for shift scheduling. Very useful introduction to the problems of call centers scheduling can be found in (Koole, 2007) and (Bloechliger, 2004). The basics of Linear Programming practical issues can be learned in (Miller, 2007).

The first possibility of shift scheduling problems is to use **Constraint Logic Programming**. In (Slim Abdennadher, 1999) the authors employed CLP to implement a system (INTERDIP) for semi-automated nurse scheduling. The computational results show that suboptimal rosters were generated for as much as 20 nurses and for one month time period in a matter of minutes. The advantage of CLP approach is the expressive power of the non-linear constraints like automaton conditions etc. The disadvantage is the limited scalability of the problem size. In general, only smaller instances of problems are tractable.

Genetic Algorithms (GA) have been used to solve rostering problems (Chang-Chun Tsai, 2008), (Margarida Moz, 2005). The great advantage of GAs is the possibility of searching vast solution sets as it can perform parallel processing, which can

effectively save processing time. The disadvantage of the GA approach is that there is no guarantee of finding optimal solution.

Column Generation (CG) is a very powerful technique for solving a wide range of industrial problems to optimality or to near optimality. It allows for solving huge linear and integer problems. In this work, however, we didn't need to use CG as the MIP approach was still sufficient for the instance of problem we were to solve. There are many papers (Jonathan F. Bard, 2005), (Brigitte Jaumard, 1997) on CG from which we were able to adopt some ideas about constraint composition.

In paper (SeydaTopaloglu, 2010) the authors apply **fuzzy set theory** to the nurse scheduling problem to treat uncertainties in the target values of the hospital management and nurses' preferences.

Linear Programming/Mixed Integer Programming approach has been used by many authors (Staffing Multiskill Call Centers via Linear Programming and Simulation, 2008), (Brunner, 2010), (Brigitte Jaumard, 1997). This approach has been widely used for the possibility of using high performance third party solvers. The disadvantage of LP/MIP is the limited expressional power of modeling nonlinear reality using linear (in)equalities. We have adopted many ideas from literature about designing standard features of shift scheduling frameworks.

A joint approach uses the **MIP, CG** and **CSP** hybrid techniques has been successfully used in (Seyda Topaloglu, 2010) planning for medical residents.

Some authors (S. Kirkpatrick, 1983) also use heuristics like **simulated annealing** to deal with multivariate or combinatorial optimization of the properties of very large and complex systems.

1.3. Outline

The paper is organized as follows. The specification of the problem is presented in Chapter 3. We describe the solved problem generally without any technical details of prospective solution approaches. Then in Chapter 4 we proceed to modeling the problem as Mixed Integer Program (MIP). We get shape to ideas from Chapter 3 using instruments of linear inequalities. In the Chapter 5 we give couple of side notes on programming issues connected to the implementation of the MIP. Then, in the last two chapters we focus on practical results and revise the meeting of objectives.

Chapter 2. Formal Description of the Solved Problem

In this chapter we are going to describe the problem we are about to solve in a formal way. Details on modeling the problem and the implementation will be object to next chapters.

Let us look at what is the ideal output of our work. Given the well-defined input we want to compute a roster that meets all the hard constraints and is optimal in terms of the optimizing criteria.

In this section we will examine the input, the participating entities and the business perspective of the problem. In the next chapter we will turn vague specification into a formal linear model.

We will proceed as follows: firstly we will analyze the important entities, their attributes and the relationships among entities. Later, we will describe all features of the rosters we are going to generate such as various requirements and limitations imposed on the collection of entities on the input. We will focus on criteria we want the solution to our problem to meet in largest extend possible. We will also discuss how these criteria interfere and what trade-offs we will be facing.

2.1. Entities, Attributes and Relations

2.1.1. Site

Site is an organizational structure of the customer's enterprise. Sites are loosely coupled with each other and we can think about them as separate entities in terms of scheduling workforce.

It may be useful to break large sites into several virtual sites as these can be more easily managed and planned for.

It is important to remember, that no entity (employees, planning periods, activities etc.) can be shared between two sites. It is of course possible to create the same activity for two sites, but these activities will be formally different.

The scope of this paper will not extend behind the border of a single site.

2.1.2. Activity

Activity is a type of work that an employee performs during a working shift. Activity can be virtually anything that the enterprise wants the employees to do during the working hours. Each employee has a set of activities that he can carry on. It is only possible to schedule an employee to work an activity that he is able to do.

Activities vs. Working Hours

Working regular activities contributes to the contract working time.

We also support special activities that are hybrids between leisure time and working shifts. These activities are scheduled, but do not count into working hours. This is necessary to support the concept of emergency. Many companies require some minor portion of the staff to be on phone and within a reach of the working site to be able to get to work in case of emergency. Such shifts are basically leisure time with some limitations so we need to treat them a bit specially.

2.1.3. Employee

Employees are the key resource. Each employee represents a real person. Hence employees can have arbitrary set of properties and preferences.

Role

There are different roles the employees can have. One employee can have multiple roles assigned at once.

The basic role is *agent*. *Agents* can be scheduled to work on activities. Another important role is *manager*, who is responsible for approving various requests, managing the general input (not based on employees) and eventually invoking the creation of the rosters.

Seniority

Some employees have earned better position in their team than other when it comes to deciding, whose requests are going to come true.

Priority on Activity

If not stated differently, we presume that all the employees are equally suitable for being scheduled to work some particular activity. However, a manager might prefer scheduling some employees on certain activities to others.

Example: In a contact center there are several supervisors who can carry out all regular activities like the rest of employees. On the top of it they can carry out the supervisor activity, which all of them should do all the time except for understaffing situation, during which they must help their colleagues out with regular activities. This is the situation, which is solved by assigning all supervisors a positive priority on the supervisor activity.

2.1.4. Contract

Each employee has a contract assigned. The contract determines the number of working days for each time interval.

The usual contracts would be full-time and part-time. The manager is responsible for the contracts being consistent with the labor-law.

It is strictly required for rosters to contain as many shifts with activities counting to contract working time for any given employee amounting to the contract-defined number working hours. If there are not enough requirements to cover the whole amount of working hours for each employee, the roster will be still generated. The output will contain notes on which employee had to be left idle and for how many shifts they were left unscheduled.

2.1.5. Working Time Preference

Most of the employees have some preferences on working hours. There are two types of preferences: shift-start and time interval preferences. The likelihood of preferences of both types going to be met is directly proportional to the priority of preference. The preferences are meant to describe the global long-term preference of employees.

Shift Start Preference

If an employee has some preference on the start of his shifts, this employee should be scheduled to work shifts that start during the period of time between start-time and end-time attribute of the preference.

Interval Preference

The other possibility is to state the time, during which an employee prefers to be at work. The desired roster for an employee will maximize the overlap of the preferred interval and the actual shifts.

Emergency

In addition to the two categories, we also have to mention a hybrid item type. These items have an activity set, but the activity is of a special type (‘not counting to contract working time’). See the next section for more details.

2.1.6. Planning Period

Planning period is a timespan during which a roster is valid. We can set up arbitrarily long planning period. However, it is practical to have planning periods of restricted length as we can forecast the absence of staff, demand for resources precisely only for limited time in future.

In practice rosters for bi-weekly to six weeks schedules are generated. Planning periods are not allowed to overlap and each and every day after the very start of our scheduling activities must fall within some planning period.

2.1.7. Resource Requirement (RR)

A resource requirement stands for a demand for a precise number of units of work on a certain activity, time and day. Resource requirements are based on business knowledge. Usually the requirements do not differ substantially among planning periods.

On each unique combination of day, time and activity there can be at most one resource requirement. If we wanted to have multiple instances of RRs with the same day, time and activity attributes, we could simply have created a single RR instance where the number of units equaled the sum over those multiple instances.

Understaffing

The resource requests have to be met in the roster. It is the most important hard constraint. However, even if there are not enough resources to cover the demand, it is useful to construct a roster that will make use of all agents available and leave as little demand uncovered as possible. Also we want to announce the manager which resource requirements were not satisfied and the volume of missing resources. Understaffing is allowed only as a last resort solution and as such is heavily penalized. Hence although technically we change a hard constraint into a soft one, we still preserve the nature of the constraint as ‘almost’ hard.

2.1.8. Optional Resource Requirement (ORR)

As seen in the Figure 1, the actual business requirements are expressed as an interval. The meaning of such interval is that the minimum values are vital to be met. If there are any resources left, use these to satisfy the requirements up to the maximal value. After analysis we decided to break the requirements into *Resource Requirements* that must be satisfied and the excess over minimal value would become an *Optional Resource Requirement*. We have also added the priority attribute to each *ORR* so that we can condition the order of satisfying the requirements. Margin for scheduling employees to work *ORRs* is between the minimal staffing enforced by *RRs* and the maximal number of man-days available in a planning period with respect to the contracts of employees.

Prio-rity	*Activity name*	*Min/Max Employees per Day*	*Start Time – End Time*	*Week-days*
-	Authorization	1-Jan	7:00-15:00	Mo-Fr
-	Operational/Authorization	1-Jan	11:00 - 19:00	Mo-Fr
-	Ticketing (Issue-Reissue)	2-Feb	6:00-14:00 a 14:00-22:00	Mo-Su
-	Queue	2-Feb	6:00-14:00 a 14:00-22:00	Mo-Su
3	VIP	2-Jan	1 agent: 9:00-17:00, 2 agents: 8:00-16:00 a 12:00-20:00	Mo-Fr
2	Operational	2-Jan	1 agent 8:00-16:00, 2 agents 8:00-16:00 a 12:00-20:00	Mo-Fr
1	Operational	2-Jan	1 agent 10:00-18:00, 2 agents 8:00-16:00 a 12:00-20:00	Sa, Fr
-	NYC	1-Jan	14:00-22:00	Mo-Fr
-	Night Shift	1-Jan	22:00-6:00	Mo-Su
-	Generic Shift		1 agent 10:00-18:00	Mo-Fr
-	Generic Shift		Any additional agents 8-12	Mo-Fr
-	Emergency	1-Jan	18:00-22:00 (in agent's leisure time)	Mo-Fr
4	Supervisor	3-Jan	1 agent 8-16, 2 agents 8-16 a 12-23, 3 agents +10-18	Mo
5	Supervisor	3-Jan	1 agent 9-17, 2 agents 8-16 a 12-22, 3 agents +10-18	Tu-Fr

Figure 2: Sample Table of Resource Requirements

Multiple Instances of ORR

Consider following business requirement about ORRs: (note: this example doesn't use the RRs from figure 2, rather comes up with new ones just for the example)

1) Satisfy all RR specified for day *D*
2) With the rest of capacity on day *D* try to satisfy as many ORRs as possible w.r.t. priority*
    a. One employee on activity A1 at 10AM with priority 10

b. One (or more) employees on A2 with at 1PM with priority 9
c. Many (five) employees on activity A1 at 10AM with priority 8

Such a requirement structure makes sense. The A1 at 10AM is an important shift that needs as many employees as possible. But we also need some for A2 at 1PM, right after we assigned some resource to A1 at 10AM.

This composition of ORRs is forbidden, as it is not directly supported by the model we have. The problem is that we cannot assign multiple priorities to one activity for some employee E1 at the same time T. It makes no difference for the employee which shift from the cases (a) to (c) he is assigned to work.

This way of specifying *ORRs* was, however, not required from the customer.

2.1.9. Employee Request

We have described the long-term preference an employee can express. But there are also one-time preferences or preferences on short periods. We will call these Employee *Requests*.

Category

Employee Requests are of two categories. First category is a *Time off* request. Time off request is an entity representing a need for absence from work.

The other category is a *Want-to-work (WTW)* request. This request type represents a wish to be at work during the period specified.

Time off Request Type

All the employee requests that belong to time off category have the request type set. This attribute expresses the motivation of the time off request. The values are self-explanatory: *Time off, Holiday, Unpaid Time off, Compensatory Time off,* and *Illness.* The set of possible request types can edited. The categories have specific properties (regulated number of requests, can count to working hours, can depend on number of overtimes worked etc.) and there is no redundancy in the set of TOR types. One TOR type needs special explanation: the *illness*. One may argue that it is not easy to hand in a time off request couple weeks before one gets sick. There are still health-connected issues that require being off-work like planned surgeries etc.

Status

Status is another property of time off requests. In general, status can be set to one of the following values: *Pending, Approved, Rejected* or *Auto*. However, in the model constructing time the requests with *Pending* and *Rejected* status were already filtered out as they cannot affect the roaster.

Rejected status means that the manager decided not to explicitly assign the employee time off nor expressed any preference on assigning the employee time off. However, rejecting a time off request doesn't prohibit the assignment of this time off. In fact there is no difference between rejecting a time off request and not requesting for the time off at all.

Approved status means that the employee associated with the actual request may not be scheduled during the span of the request.

Auto status means that it is up to system to decide if there is the enough overcapacity for approving that request. Approving requests set to auto is of course preferred.

Pending the initial status of newly inserted request is *pending*; it can be changed to any of the preceding status values; for the solver, no requests with *pending* status are considered.

Contract-Working-Time (CWT) Component

The *CWT Component* attribute is important when it comes to meeting the requirements on contract working time. Some of the requests, like holiday, count as working days with respect to contract based working hours, whereas requests like *unpaid time off* do not. The CWT Component attribute determines this.

In case of time off requests, if the CWT component attribute equals true and the request is to be satisfied, there will not be scheduled any shifts on any day that had some overlap with the request.

2.1.10.Employee Request Policy

Some employees may be taking great advantage of the possibility of claiming unlimited number of requests and preferences. We want to be sure that the generated rosters will not suffer from extreme distortions due to exaggerated preferences of individual employees.

Restrictive Policy

The first way how to make the requests fair is to set a limit on number of requests per planning period for each employee.

Adaptive Policy

Another way is to assign each request a special weight that derives from the total number of requests the employee has handed in.

We can combine this technique with the restrictive one in two ways.

We can set two thresholds:

- Lower limit – number of 'free' requests. If the total number of requests does not exceed this limit no weight of any request will get decreased
- Upper limit – the maximal number of requests per employee per planning period (as above)

Priority

The priority attribute of an employee request determines the chance that the request will get approved by system, if it has the status set to auto. The priority is relative to other requests of a single employee.

2.1.11.Roster

Roster is the final product of the workforce scheduling. It is basically a set of roster items that belong to a specified planning period. A roster may be displayed as a matrix where rows stand for employees and columns stand for days. Then the roster items are placed in the cells of such matrix.

Different choice of rows and columns may as well make sense.

Note that some cells of the matrix may be left blank. Such places stand for leisure time.

See a sample roster generated during testing phase on the next page.

Published Roster

In practice, rosters are usually generated in the middle of a planning period. After the manager is satisfied with the roster quality, the candidate roster will be *published*. Once published, the roster becomes official and is presented to all employees involved.

Therefore only minor changes should be performed on a published roster as all the employees already count on it. Usually the manager ends up changing an already published roster because of unpredictable situations like illnesses, emergency situation etc.

2.1.12.Roster Item

As mentioned above, a roster comprises of a set of roster items. Each roster item naturally binds with an employee and has a start time and duration. The rest of the attributes depends on the type of roster each individual item. The roster items are of two types.

Activity Roster Item

The activity roster item contains activity field. These items stand for regular working shifts.

Time off Roster Item

Some of the time off is explicitly marked in roster. This happens, when there was a request for some kind of time off that was satisfied. We store the type of request with the roster item to be able to tell later, what the employee asked for.

2.1.13.Entities and Relations

The Figure 4 shows the main entities participating in the solved problem as well as shows the relations among some entities.

Valtrová Eva	S9	S9	S9	S9	V	S9 S14	S8	S12		D	D	V	V	D	D	D	D	D	V	V	S8	S9	S9	S9	S12	V	V	S12	S9	S9	S9
Olšovská Alena	S9	S9	S10	S8 S8	V	V	S10	S10	S9	S7 S7		V	V	S10	S10		S10	S9	S9	S9		S10	S10	S10	S9	S9		S10	S10	S10	S10
Marková Eliška		S10	S8	S9	V	V	S12	S9	S10	S9	S9	V	S9 S8	S8	S9	S9	S9	NV	V	V	S10	S12	S12	S12	S10		S9 S8	S8	S12	S12		
Dubová Eva	S11	S14	S14	S14	S8		S7	S14	S12	S8 S8		V	V	D	S11 S11	S14	S14	S14	S10 S8	V	S7	S14	S14	S14	S14	V	V	S12	S12	S14	D
Hausnerová Lenka	S14 S14	S11 S11	S7 S7		S6 S8	S6 S6		S10 S10	S7	S6 S6	S7 S7	V	V	S11 S11	S14 S14	S10	S8	S7 S7	V	V	S11 S11	S8	S6	S9 S9	S6 S6	V	V	S14 S14	S11 S11	S9	S6 S6
Honzová Nikola	S6	S6	S9	S7		S6	S6	S9	S9		S8	S22	S22	S22		S6		S6	S12	S10	S6	S10		S22	S22	S22		S10			S14
Jerábková Anna	S8 S6	V	V	S12 S12	S14 S6	S14 S14	S11	S8	S6 S6			S6 S6	S6 S14	S7 S7	S22 S22	S22 S22			S6 S6	S14 S14	S12	S9	S7 S7	S6 S6		S6 S6	S6 S14	S6 S6		D	D
Kolárová Magdalena	S7 S6	S8	S6	S6 S6	V	V	S6 S	S6 S6	S10 S8	S22 S22	S22 S22		S9	S6 S6	S6 S6	S12	S11	S8 S	V	V	S6	S6 S6	S14	S12	S8 S6	V	V	S11 V	S8 S6	S12	S11
Mikesková Anna	S14 S6	D	D	D	V	V		S22 S22	S22 S22		S11 S10	S6	S6 S6	S10 S10	S7 S7		S6 S6	S14 S10	S14 S14	S14 S14	S10 S14	V	S10 S	S7 S7	S9 S8	S6 S6	S6 S6	V	S14 V	S12 S6	S8 S8
Milevová Katarina				S14 V	S14	S12	S14	S11		S14	S14	S12	S10		S12	S12	S22	S22			S12	S12	S14	S14	S14	S10		S7	S14	S14	S14
Myloslavska Olga	S22		S14	S11	S14	S14 S14	S14	S14	S11		S6	S14 S14	S14 S14	S14	S14	S11		S14	S14 S14	S14	S14	S14	S14	S11		S14	S14 S14	S14	S14	S11	S7	
Priadková Michaela		S14 V	S14 S14	S14	S10		S14	S14	S14	S11		S10	S14	S12	S9	S8	S6 S6		V	V	S14 S14	S11	S8	S14	S11		S12	S9	S7	S6 S6	
Pytelová Lucie	N	N	N	N			N	N	N	N	N			N	N	N	N	N			N	N	N	N	N			N	N	N	N
Rubášová Lucie	S14 S14	S14 S14	S11 S14		S12 S12	S10 S8	S9 S9	S7 S7	S8 S8	S9 S9		V	V	S14 S14	S14 S14	S14 S14	S14 S14		V	S12 S12	S14 S14	S14 S14	S12 S12		S10 S10	S8	S10 S8	S6 S6	S6 V	S8 S8	S6 S6
Sainerová Jana	NV	S22 S22	S22 S22	S22 S22	S22 S22	S22 S22	S22		S14	S14	S14	V	V	S14	S12	S9	S7		S22	S22	S22	S22	S22			V	S22 V	S22 S22	S22 S22	S22 S22	S22 S22
Špánová Nadežda	S6	S7 S7		S6 S6	S6 S6	V	S8 S8	S6	S6 S6	S6 S6	S6 S6	V	V	S8 S8	S8	S7 S7	S9	S6 S0	V	V	S8 S8	S6	S6 S6	S8	S6 S6	V	V	S8 S8	S6	S6 S6	S9
Turecková Kamila		D	D		V	V	D	D		S14 S14	S14	S14	S12	S9		S14 S14	S12	S9	S6 S6	S6 S6		S7 S7	S12	S11	S7 S7	V	S14 S14	S14 S14	S14	S14	S14
Vágnerová Hana		S6 S6	S6 S6	V	V	V	S22		S14	S12	S9	S6		S6	S6	S6	S14	S11		S6	S9	S12	S9	S6		S14	S14	S12	S9	S11	S7

Figure 3: Real Roster

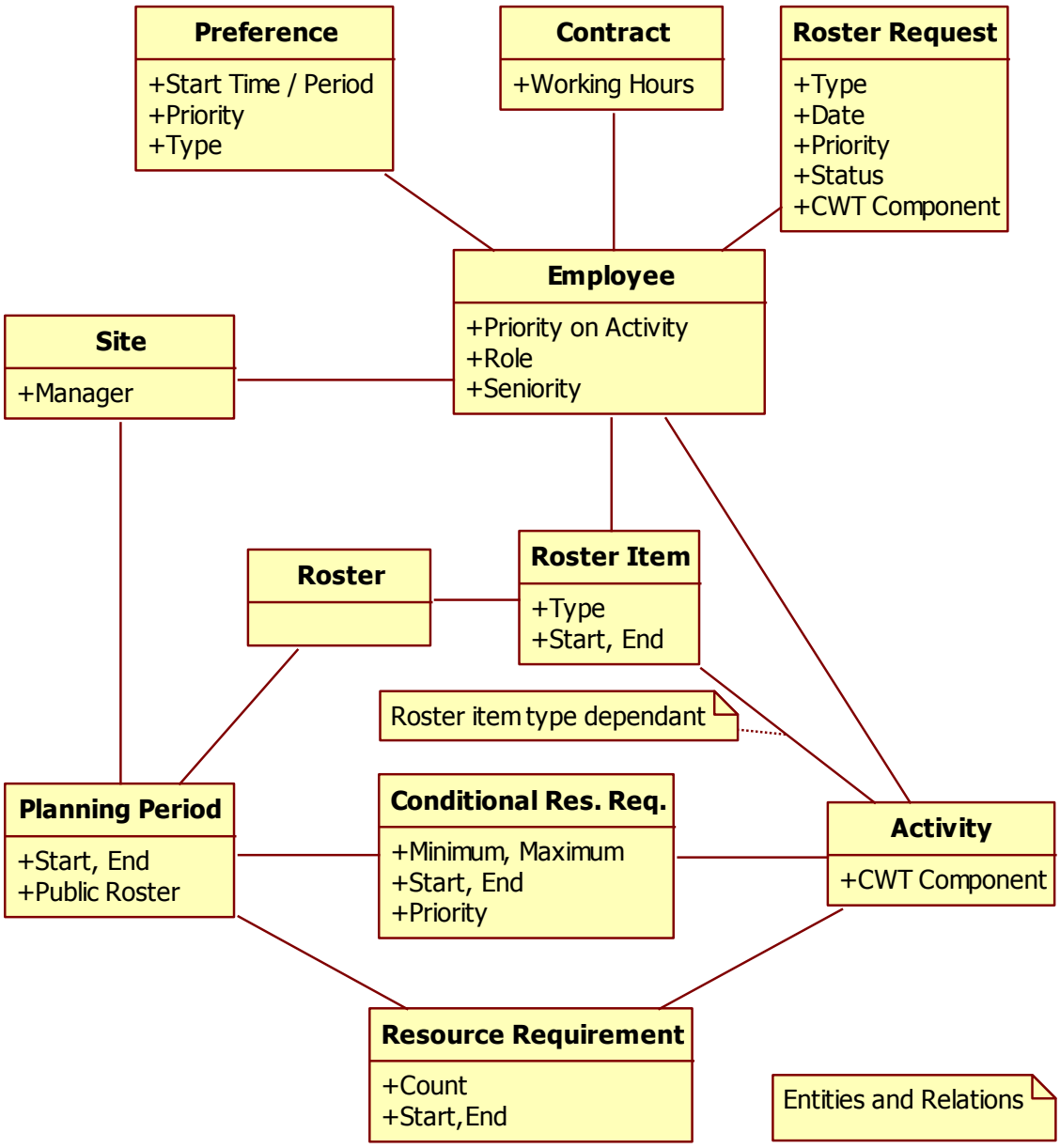


Figure 4: Entities and Relations

2.2. Roster Features

There are a number of various limitation and constraints to the roster. There are also various qualities we need the roster to have.

We have already described many features during presentation of entities. In the following part of the chapter we will describe what has not been mentioned yet.

2.2.1. Pre-made Roster

It is possible to leave the whole construction of the roster to the system. Nevertheless, managers often want to assign some activities on certain days to certain employees. In general we let the manager to construct the roster himself to whatever stage of completeness. This pre-made roster will then serve as yet another input.

There are not the same rules applied to the pre-made roster items and to the ones we are going to generate. Some conditions don't need to hold when it comes to manager-inserted roster items. Our job here is to generate a valid roster with respect to all the restrictions mentioned in this chapter; nevertheless there are situations in practice, where the manager needs be able to override virtually any limitation to be able to solve some critical situations. We will mention these cases when describing the individual features that are affected by the non-standard composition of roster items coming from the pre-made roster.

Matching Input Roster Items

Let us now consider only roster items from the pre-made roster that have some activity set. These roster items may or may not correspond to some *(optional) resource requirement*. If a roster item has a matching *(O)RR*, we want to make sure that the roster items from the input roster will be paired with corresponding *(O)RR* if possible.

After having found a matching (O)*RR* we lower the number of required employees on the given activity and time. If the number of required employees drops down to zero, we discard the requirement. If this pairing hadn't been done, we would have ended up with shifts that were not required. More precisely we could schedule up to twice as many employees than required on some shifts.

2.2.2. Inter-Shifts Breaks

The Czech Labor Code states, that an employee must have at least twelve hours of uninterrupted rest between any two consecutive shifts. The regular twelve-hour break can be reduced all the way down to an eight-hour break. In such case, the following break must be extended by the portion of time that was removed from the previous break. Hence, the break can be extended up to sixteen hours.

We will relax this constraint in two ways. Firstly, we will not be utilizing the possibility of shrinking any twelve-hour break in case of roster items generated by the system.

Secondly, we will allow the manager to place the adjacent roster items in the pre-made roster arbitrarily close to each other – the break can be reduced to zero length. If the break reduction exceeds four hours (as is defined in the Labor Code) the following break will only get extended by four hours.

For completeness let us mention, that it is forbidden for two adjacent roster items for one employee to overlap.

The following figure shows the typical situation.

Figure 5: Standard Inter-Shift Break

The next figure shows the reduction of breaks between shifts.

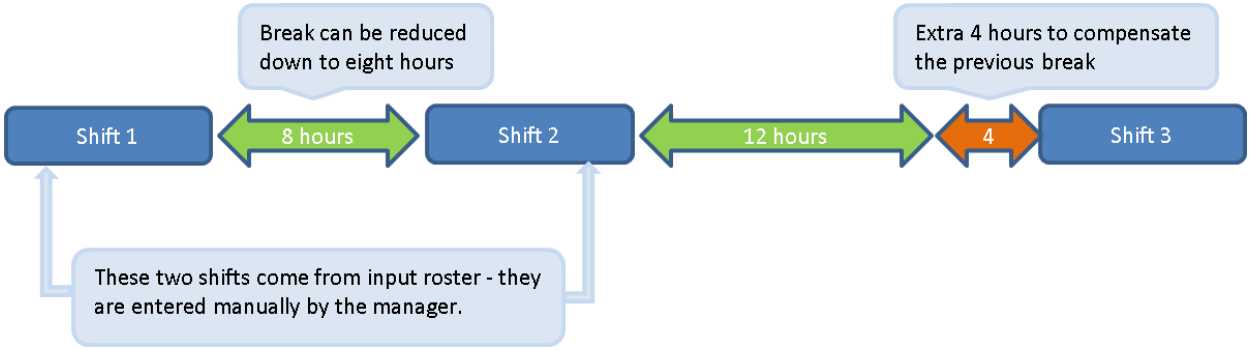


Figure 6: Extended Inter-Shift Break

2.2.3. Fair Policy

One of the important features of a roster is that for it does not seem to be unfair for some employees. We have spent substantial amount of time on working with various approaches of making the roster fair. It shows that there cannot be any ultimate solution to fair policy problem because of all the trade-offs between employee preferences and fairness.

We have identified two loosely coupled categories of fairness. We want to make the activity distribution over employees possible evenly distributed as well as the shift distribution (with respect to start-time).

In short, the idea of fair policy is to compute an expected number of shifts (hours) that an employee should be working some activity. We want to minimize the difference between the expected value and the actual number of such shifts.

Employee-based Fair Policy

The first idea is to minimize the difference for each employee. This works nice, but there is one serious drawback. The following example documents the problem:

Example FP1: We have six employees and 30 resource requirements demanding one employee each that is 30 man-days. Clearly, the expected number of shifts for an employee is 5, thus a perfectly fair assignment would be (5,5,5,5,5,5). We may not be able to achieve such assignment for various reasons (other activities, employee requests etc.).

One still quite even assignment would be (4,4,4,6,6,6). The total deviation from expected values is 6. Now the problem is that one very distorted assignment of (2,5,5,5,5,8) has total deviation also 6. It is clear, that the employee with 2 and 8 shifts would not accept the assignment as fair.

General

If we want to fix the problem from last example we need to minimize the maximal deviation.

Example FP2: The two assignments (4,4,4,6,6,6) and (2,5,5,5,5,8) have the same total deviation from expected value equal to 6. However, the maximal deviation in the first case is equal to 1 and in the second case equals 3. This way we can give priority to the intuitively more fair assignment.

It may not be sufficient to minimize the maximal deviation. The problem may be recur

Example FP3: Let us continue with the example FP2. It may be the case, that the first employee can only work two shifts of the given activity because some other limitations.

Under such condition, the assignments (2,5,5,5,5,8) and (2,2,5,5,8,8) are equal in terms of maximal deviation.

We would have to come up with second greatest deviation and recursively proceed to next levels. It shows, nevertheless, that in practice the single level is enough.

2.2.4. Weekly Rest

During any given seven consecutive days, an employee has to have a thirty-five hours long break. No scheduled activity should interfere with this break.

The only exceptions are activities that do not count to working hours (like emergency). These activities do not require employees to be physically present at work, rather to be on the phone and ready to arrive to the office in case of emergency.

Past Roster Issues

The requirement for weekly rests has to be satisfied across all rosters. Usually the planning periods are longer than seven days. Therefore the condition of giving the employees weekly breaks virtually has to be enforced just on the border between the two adjoining planning periods (rosters) in addition to the breaks within the actual roster to be generated.

Thus to construct a roster we need information about previous rosters.

2.2.5. Limits on Activities

Manager can restrict the number of shifts an employee is going to be scheduled on some activity. It is possible to set an upper bound, lower bound, both or none. This feature can potentially become dangerous. There are two ways things can go wrong here.

Firstly the manager can set the lower bound too high. In such case there may not be enough requirements to support the minimal number of shifts that the employee is required to work. The other extreme case is that the manager sets too low limits on

some activity for all employees. Then there are may not be enough resources left to cover the resource requirements.

It is solely up to the manager to be aware of the possible consequences of setting the limits on activities for employees too tight as it may lead to infeasibility of the rostering problem.

To some extend it is possible to check the total supply and demand on various activities statically before generating the roster. But it is possible to wind up with infeasible problem even if there is a slight overhang of demand.

2.2.6. Employee-Activity Priority

For any combination of employee and activity it is possible to set a priority of scheduling the given employee on the given activity. The reason for setting these priorities is usually the level of skills of an employee. We want an experienced employee to work on activities that are either crucial or need to be done right. On the contrary we might want the newcomer to be working on non-critical activities preferentially.

On the top of this criterion, some employees simply prefer one activity to another.

2.2.7. Previous Roster(s) Consistency

Planning periods are no isolated periods in time. It is very important that the roster we are going to generate is compatible with the previous roster or even a series of preceding rosters.

Many of the roster features described have to work not only in the actual roster but also on the border with a previous roster. Also all the following rosters will have to be compatible with the roster for the current planning period.

Note, that there may be multiple rosters for each of the preceding planning periods, but just one can be *published*. Only roster items that matter to the algorithm are those from *published* rosters. For various features a different time-span of past planning periods is relevant.

Let's look at the features influenced by the past roster items.

Weekly Rest

We will need to extend the borders for the sliding seven-day time window six days ahead. We have to assure that all employees will be guaranteed a regular weekly break after even if a portion of the work was done before the actual planning period.

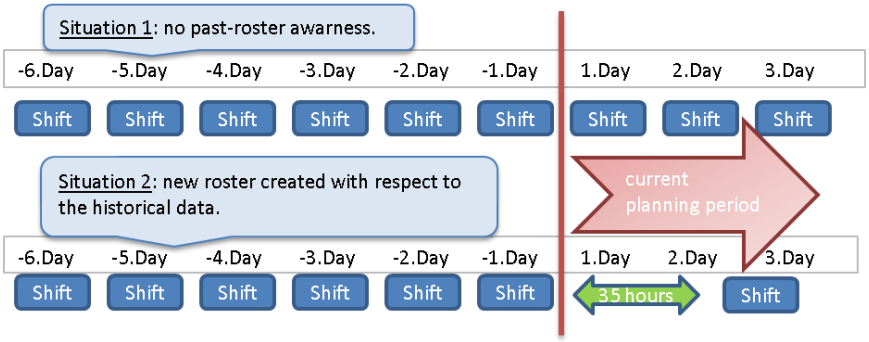


Figure 7: Boundary Situation – Weekly Rest

Inter-Shifts Breaks

When we want to schedule a shift to an employee, among other things we have to check when the latest end of a scheduled shift in the last planning period occurs. Then we have to enforce the necessary twelve-hour break that has to be present between two consecutive shifts as described earlier. Some roster items may even overflow into the current planning period. We must be aware of all these marginal situation.

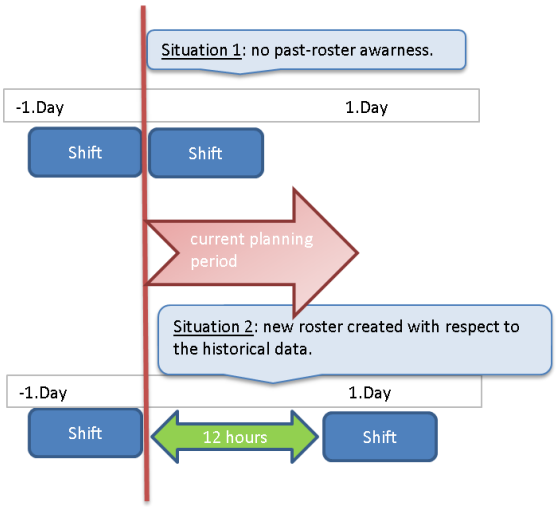


Figure 8: Boundary Situation – Inter-Shift Break

Employee Requests Fairness

The preference policy described earlier in this chapter contains limits and weights that can be altered according to the number of requests stated in the past planning period.

We could compute the number of requests with status *'auto'* that were satisfied and use the request-satisfaction ratio in the actual planning period to slightly increase the chance of satisfying requests of employees with ratio lower than average and also

decrease this likelihood for employees with higher ratio. In the long run we could achieve some fairness in satisfying employee requests this way.

Fair Policy Carry-Over

The similar way as we have described the carry-over in case of employee requests we can alter the expected number of shifts for each employee on various activities according to the shift composition in the last periods.

2.2.8. Employee Time Preference (ETP)

Some employees prefer to start working early; some don't like working night shifts. We support the option of specifying the preferred working hours. This is a long-time preference that is taken into account for each shift generated during any given planning period.

The preference is a *real* value from the interval [-1; +1]. The higher the value, the more the employee prefers the specified shift. Analogously the lower the value the less the employee likes the shift. Although the domain of preference is an interval on real axis, employees always tend to pick very limited number of values – typically the set {-1, -0.5, 0, 0.5, 1}.

The step for the requests is one hour. Each employee may theoretically set up to 24 preference instances. If no record for some start-time is present for some employee the employee is supposed to be indifferent.

If an employee wants to specify a one-time preference, the *Employee Requests* will serve this purpose. An ETP can be specified like a triple (start hour, employee, preference).

2.2.9. Time off Distribution

Another aspect that can be of value is the distribution of days-off. It can be very frustrating to have the days-off randomly scattered all over the schedule with no intention of grouping them into larger blocks. Employees naturally make more use of two or three days-off in a row than the pattern of a day at work, day-off, day at work, day-off etc.

After deeper analysis we have identified two different approaches.

Grouping the Days-off

The aim here is to generate blocks of leisure time of maximal length. In fact the aim here is to create the maximal number of two consecutive days-off.

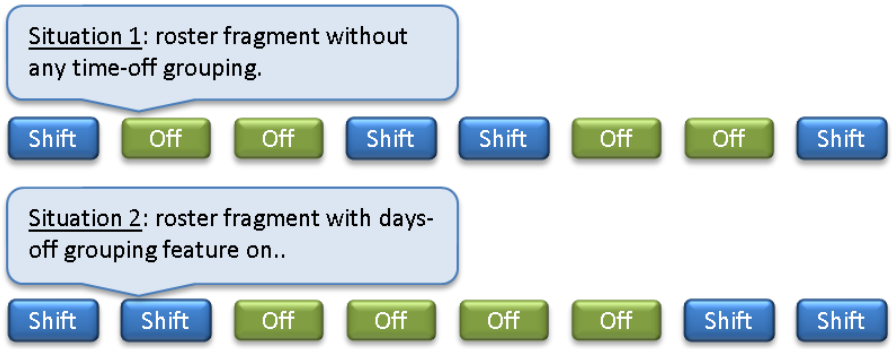


Figure 9

Penalizing Single Day-Off

The rosters with the *day-off grouping* tend to have long blocks of leisure days. Let us look at following situation:

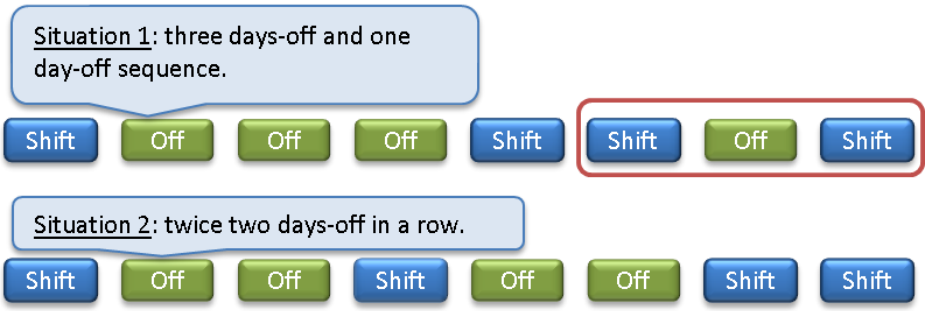
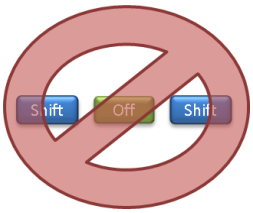


Figure 10

The idea here is just to penalize the single day-off. All other combinations are allowed.



There are four days-off in both cases. Both the situations are equal in terms the *days-off grouping*. We have managed to have two couples of leisure days there.

During the analysis we have discovered that it is much more important to prevent the rosters from containing the *single day-off* pattern than piling up the days of. In the

latter section we will examine the practical results and compare the two approaches again.

2.2.10.No Emergency after Six Shifts

The need for this feature might not be obvious at first sight. We want to forbid the following sequence:

Figure 11

Firstly, the emergency is a day-off in general. The employee scheduled to keep the emergency is required to be on the phone and within the reach. It is not very likely, but in case of emergency this employee has to come to work.

In general case after being called up the manager cancels the next day's shift and gives the employee that served the emergency a day-off.

The problem here is that any amount of leisure time after the emergency can fulfill the *weekly rest* condition. In case of emergency the employee keeping the emergency shift would have to work seventh day in a row which would be illegal.

2.2.11.No Night Shift after Emergency

Yet another possible complication with emergency can occur. In the real-life problem we are solving there is usually just one employee on the night shift. As described above, if an employee is scheduled on an emergency, it may happen that the very next shift of that very employee is cancelled. The rest of employees are able to make up for a single missing employee for some short time (one shift). However, if one employee is scheduled for emergency followed by night shift, there is no-one left working the night shift to cover the missing resources. The only possible solution would be to use the employee that is on emergency. This would however trigger a cascade of call-ups. There is a good chance that in a long-enough chain of call-ups one employee would be unavailable and the night shift would be left unhandled.

2.2.12.Contract Working Time Enforcement

The maximal number of shifts an employee can be scheduled to during a planning period is determined by a working contract. On the other hand the employer wants to use all available resources. If all the *RRs* are satisfied, there is a set of *ORRs*. The list

of *ORRs* should contain demand for resources of sufficient volume to employ all employees to the extent of the contract working hours.

Anti-Dummy Agents

If it is, however, impossible to create a roster where each employee is scheduled to as many shifts as the contract allows, we may allow for the creation of such the roster. As a part of result besides the actual roster, there would be a number for each employee saying how many days there are left with respect to contract working time. Of course the number of hours left minimized each agent. But if it is impossible to create a roster of perfect CWT coverage, it will allow the manager to get a closest approximation of it and correct the discrepancy manually by adding some special shift for instance.

2.3. Objectives Overview

Substantial differences in the price of a unit of work for different employees can be often observed. It is frequently the primary target to optimize the total cost of the call center personnel, often by finding the minimal staffing levels and firing all the remaining staff.

In the condition of this very problem it isn't, however, the main aim, nor is it expected to be done by the software we are going to design at all. The customer is willing to keep the staffing limits as they are, mostly because it requires quite some time as well resources to train the agents of the level of proficiency needed.

The pricing in this case would be the degree of satisfaction various user requests and preferences as well as using the maximal number of resources. The preferences stated by management of the company will be also optimized. Last, but not least, if some hard constraints get relaxed to soft constraints (by management), the violation of these constraints will be minimized too.

Chapter 3. Mixed Integer Model

In the previous chapter we have described the problem in a formal way. There are multiple ways of solving the problem. One of the most promising approaches is the linear model with binary decision variables. The model we are going to design is often called in literature as *implicit linear model*.

Firstly we will define some basic terms. Then we will show representation of entities, relations among them and roster features in the model. Lastly we will put all bits together in the model-recap section.

3.1. Mixed Integer Program

Let us start with some definitions. A mixed integer program (MIP) is defined as follows. See (Achterberg, 2007).

Definition (mixed integer program).

Given a matrix $A \in \boldsymbol{R}^{m \times n}$, vectors $b \in \boldsymbol{R}^{m}$, and $c \in \boldsymbol{R}^{n}$, and a subset $\boldsymbol{I} \subseteq \boldsymbol{N} = \{1, \dots, n\}$, the *mixed integer program* MIP $= (\mathrm{A}, \mathrm{b}, \mathrm{c}, \mathrm{I})$ is to solve

$$(\mathrm{MIP}) c^{*} = min\{c^{T}x \mid Ax \leq b, x \in \boldsymbol{R}^{n}, x_{j} \in \boldsymbol{Z}\ for\ all\ j \in I\}$$

The vectors in the set $X_{MIP} = \{x \in \boldsymbol{R}^{n} | Ax \leq b, x_{j} \in Z\ for\ all\ j \in I\}$ are called *feasible solutions* of MIP. A feasible solution $x^{*} \in X_{MIP}$ of MIP is called *optimal* if its objective value satisfies $c^{T}x^{*} = c^{*}$.

The bounds of the variables are denoted by $l_{j} \leq x_{j} \leq u_{j}$ with $l_{j}, u_{j} \in \boldsymbol{R} \cup \{\pm\infty\}$. Formally, they are part of constraint system $Ax \leq b$, but in practice they are treated outside the coefficient matrix. Depending on the integrality status and the bounds of the variables, we define the following subsets of the variable indices N = {1,…,n}:

binary variables : $B \coloneqq \{j \in I \mid l_{j} = 0\ and\ u_{j} = 1\}$

integer variables : I

continuous variables : $C \coloneqq N \setminus I$

Specializations of MIPs are

- *linear programs* (LPs) with $I = \emptyset$
- *integer programs* (IPs) with $I = N$
- *mixed binary programs* (MBP) with $B = I$, and
- *binary programs* (BPs) with $B = I = N$.

If we remove the integrality restriction from an MIP, we obtain its *LP relaxation*:

Definition (LP relaxation of an MIP).

Given a mixed integer program MIP $= (\mathrm{A}, \mathrm{b}, \mathrm{c}, \mathrm{I})$, its *LP relaxation* is defined as

$$(\mathrm{LP})\check{c} = min\{c^T x \mid Ax \leq b, x \in \boldsymbol{R}^n\}$$

$X_{LP} = \{x \in \boldsymbol{R}^n | Ax \leq b\}$ is the set of *feasible solutions* of the LP relaxation. An LP-feasible solution $\check{x} \in X_{LP}$ is called *LP-optimal* if $c^T \check{x} = \check{c}$.

The solution set of the LP relaxation defines a polyhedron $P = X_{LP}$. This *LP polyhedron* is a superset of its integer hull $P_I \subseteq P$, which is the convex hull

$$P_I = conv\{P \cap (\boldsymbol{Z}^I \times \boldsymbol{R}^{N \setminus I})\} = conv\{X_{MIP}\}$$

of the MIP feasible solution.

Definition (Constraint)

Any row of the system of (in)equalities defined by the matrix *A* is a constraint. Each constraint we will include into the model will have some meaning which will be presented throughout this chapter. Total number of constraints is *m (=nr. of rows in the A matrix).*

3.2. Constraints Classification

There are a number of limitations of what employees are allowed to do and there are also many things that employees would like to do within these limitations. Let us look at the nature of these limitations and wishes that will play an important role in formulating our problem and categorize them first and then take one constraint at a time and define them formally. A very nice overview of constraint categories can be found in (Bloechliger, 2004)

3.2.1. Hard and Soft Constraints

The first distinction is a very natural one. A ***hard constraint*** is a constraint that always has to be satisfied in order to gain a valid solution.

Example: every employee has to work at most eight hours a day.

A ***soft constraint***, in general, expresses a sought-after quality of a solution, which doesn't need to be fulfilled completely. It still can be required to satisfy the constraint to some threshold level.

Because fulfilling the requirement expressed by a soft constraint is favorable, we need to appreciate solutions having this quality and penalize solutions lacking it. Each soft-constraint can have a different priority.

If it is the case, that the soft constraint will be enforcing meeting the requirement at least on the threshold level, this will result in a classical hard constraint, as mentioned above. This part of soft-constraint is optional.

Example: we might want an employee to start working at the same time each and every day in the generated schedule. But it would be satisfactory, if the time-span of the earliest and the latest start of the working day is four hours.

3.2.2. Local and Global Constraints

A constraint is asserting some condition on some of the entities describer earlier in this chapter. Some of the conditions only take into account handful of adjacent entities (say couple of shifts of the same employee, activity or shifts during some time interval which spans over couple of days). Such conditions are called ***local***, because their satisfaction only relies on states of entities that are 'close' to each other in some way. An example can be: *Each employee must have twelve hours break between shifts.*

Some other conditions rely on entities that spread across the whole variable problem arbitrarily. We call this class of constraints ***global constraints***. An example can be: *During the planning period an employee cannot work more shifts in total than it is allowed by his contract.*

3.2.3. Constraints by Origin

There are different reasons for introducing constraints into the model. Constraints can rise implicitly from the nature of the problem itself, some constraints reflect business goals. A good example is constraint requiring some specified number of employees working at specified time on some activity. Labor code is another prolific source of restrictions that need to be incorporated in the model. In this category we would list inter-shift breaks between following shifts, rules for assigning days off, maximal number of shifts per month, shift length etc.

3.3. Variables

In this part we will define entities and terms from which the whole scheduling problem constitutes. The building stones of the model are activities, shifts, employees and (time) periods. The binding elements on the input side are *resource requirement*s and *optional resource requirement* that require employees to be scheduled on some particular times and days to work activities specified. The output are shift instances with specific activity and time allocation assigned to employees.

In the general definition of MIP, we have just mentioned, is the set of variables $\boldsymbol{N} = \{1, \dots, n\}$. When we model a problem as a MIP, different variables can have different meanings. In fact, there are several categories of variables with distinct semantics. Domains of variables from different categories can differ too. In the formal definition of MIP written before, we were working with the vector *x* representing all variables in the model. For the sake of readability we will be using more intuitive notation for variables exploiting various indices and variable labels.

When we are creating the model, we encode all relevant information implicitly into the model. However, it is not possible to find out the meaning of some arbitrarily chosen variable. In other words the implicitly encoded information is not explicitly accessible just from the variable's label. This is especially critical when we want to decode the meaning of the (optimal) solution of the MIP, we obtain from the solver. Therefore we need to store the explanatory information about variables during the model composition, when the information is available. We will call this information *meta-data* information. We will be accessing the values of the meta-data attributes through the dot notation (e.g. X.attribute).

The domain of binary variables consists of two values. We will call one of the values *true* or *1* and the other *false* or *0*. We will use the *true/false* and *1/0* notation interchangeably so that it fits into the context as much as possible.

Also we may write that we set or assign some value $\boldsymbol{c}$ to a variable $\boldsymbol{X}$. Such statement is nothing but a technical shortcut for introducing a unary constraint tightening bounds on that very variable to a single value ($\boldsymbol{c} \leq \boldsymbol{X} \leq \boldsymbol{c}$).

We might end up in a situation, when we find out, that some variable is not needed in the model, but it already has been included. In such case we simply tighten the bounds on that variable.

In the next part we will describe types of variables that we consider in the model.

3.3.1. Roster Item Variables

Variables of this category express some possible allocation of resources. As such it bares *Employee, Activity, Date, Start Time* and *End Time* self-explanatory meta-data attributes. If the variable is set to true, it is equivalent to the employee *e* working on day *d* at time *t* on activity *a*.

The attribute *Counts to CWT* determines, if the shift corresponding to the roster item variable counts to the contract specified working time or not. The value of this attribute is derived from the *Activity* attribute, as it is stated for each activity, whether shifts of the activity count to working hours or not.

There are several ways a *roster item variable* can be added to the model: The first way is natural one – through demand satisfaction. Recall the *resource requirements* and the *optional resource requirements*[1]. There are many requirements for resources each day. For each such requirement we find all employees that can work the required activity during the specified time period. Then we add one roster item variable for each employee from the list of eligible employees. If a roster item variable is based on an *optional res. Requirement* then the *is ORR* attribute is set.

Second way leads through the *premade roster*. If an employee has already been assigned some shift by the manager (in the *premade roster*), we need to create a corresponding roster item variable. Then the flag attribute *is From Premade Roster* is set for the variable.

Last way of introducing a roster item variable is based on the past rosters. For each roster item from the past roster(s) we consider (see the *previous roster(s) consistency* section) we also need to create a corresponding roster item. In such case the attribute *is From Past Roster* is set for the variable. When the variable is based on a roster item from either the *premade* or some *past* roster, the value of such variable is set to one.

[1] These requirements contain reference to activity, time interval, day and number of employees needed during the specified period.

Meta data attributes overview: Employee, Activity, Is-ORR Flag, Date, Start Time, End Time, Counts to CWT[2] , Is from Past Roster, Is from Premade Roster.

Notation:

- $X_{e,a,d,t}$ for Resource Requirement based variables
- $\mathcal{X}_{RR}$ is the set of roster item variables based on resource requirements
- $\mathcal{X}_{ORR}$ is the set of roster item variables based on optional resource requirements

Domain: {0,1}

$$X_{e,a,d,t} \begin{cases} = 1 \; if \text{ employee } e \text{ works activity } a \text{ on day } d \text{ at time } t. \\ = 0 \; otherwise \end{cases}$$

3.3.2. Time off Variables

Time off variables are a complement of the roster item variables. A time off variable is a binary variable that is set to *true* if the corresponding employee is not scheduled to work during a specified day *d* (i.e. there is no roster item variable set to *true* that would overlap with the day *d*). This variable can be seen as syntactic sugar, but it reduces the complexity of some constraints. It clearly follows that *time off variables* bare the meta-data information about the *Employee* and *Day* they are bound to.

Meta data attributes overview: Employee, Date.

Notation: $F_{e,d}$

Domain: {0,1}

$$F_{e,d} \begin{cases} = 1 \; if \text{ employee } e \text{ doe not work on day } d \text{ .} \\ = 0 \; otherwise \end{cases}$$

3.3.3. Dummy Variables

Some of the constraints we will be presenting are hard constraints in their nature. However, from the business point it is not acceptable to gain an infeasible problem due to the impossibility of satisfying these constraints. The idea is to add special (dummy) variables to the constraints that will fix the feasibility of the very constraint in an extreme case. To assure that the original functionality won't be changed in general 'feasible' situation, we set very high prohibitive rate to this variable in the

[2] In the case of our customer this flag is used to describe the exceptionality of the emergency activity; we need to handle it alternatively when constructing some constraints, yet we need to include it in the list of activities.

objective function. (i.e. in the cost vector *c*). There will be several constraint-specific sets of these special variables used for different purposes throughout the model. We will use special variables to allow for violating the constraints enforcing following roster qualities: the satisfaction of resource requirements ($\boldsymbol{X'_{a,d,t}}$), approved want-to-work requests ($\boldsymbol{Xw'_{e,d,t}}$), employee shift/activity limits ($\overline{ES'_{e,t}}$, $\underline{ES'_{e,t}}$/$\overline{EA'_{e,a}}$, $\underline{EA'_{e,a}}$).

Metadata: There is no common metadata for all classes of dummy variables; the possible attributes are *activity*, *day*, *time*, *employee* and *shift.*

Notation: $\boldsymbol{X'_{a,d,t}}$, $\boldsymbol{Xw'_{e,d,t}}$,$\overline{EA'_{e,a}}$,$\underline{EA'_{e,a}}$,$\overline{ES'_{e,t}}$,.$\underline{ES'_{e,t}}$.

Domain: [0, +inf)

3.3.4. Helper Variables

Not all of the variables used in the mixed integer linear problem have some direct interpretation. Sometimes we need to introduce helper variables for technical reasons. In mixed integer program there are no other constructs allowed than linear expressions. Sometimes it is necessary to use absolute values, logical operators, maximum or minimum from some variables etc. To be able to use these amenities we usually have to come up with helper variables to emulate these functions. For details on linearizing various formulas and function see (Miller, 2007). Some helper variables are used for technical reasons, however, they do not emulate any concrete well-known function.

Metadata: There is no common metadata for all types of helper variables; the possible attributes are *employee*, *day*, *activity*, *time* and *break type*.

Notation: $\boldsymbol{DoG_{e,d}}$, $\boldsymbol{DoG'_{e,d}}$, $\boldsymbol{T^t_{e,d}}$, $\boldsymbol{Y_{e,a}}$

Domain: **R** or **N**

3.4. Objective Function

Substantial differences in the price of a unit of work for different employees can be often observed. It is frequently the primary target to optimize the total cost of the call center personnel, often by finding the minimal staff levels and firing all the remaining staff.

In the condition of this very problem it isn't, however, the main aim, nor is it expected to be achieved at all. The customer is willing to keep the staffing limits as they are, mostly because of the costs and length of employee training.

As a matter of fact, we maximize the fulfillment of employees' preferences, priorities and requests. The resulting roster endeavors to comfort the needs and wishes of all employees. Technically, we minimize the penalty for not fulfilling preferences, requests and not respecting various priorities.

Implementing roster features is a complex issue. Very often we need to both introduce new constraints and add new expressions to the objective function in order to implement some new feature. That is the reason why we do not describe the objective function in one place. We rather introduce its bits when describing implementation of individual features. Nevertheless we conclude this section with a complete model recap where we present the composition of the objective function without further comments.

3.5. Implemented Constraints

In this section we will go through each constraint type implemented in the system. We will show the implementation of the roster features described earlier. We will state the (in)equalities defining each constraint in the mixed integer program as well as describe the meaning of it. We will discuss the contribution of each roster feature to the objective function too.

3.5.1. Demand Constraint

This is a vital constraint that actually enforces assigning non-zero values to decision variables. The input for this constraint is a *Resource Requirement (RR).* A resource requirement has these attributes – activity, date, time and count.

The set of variables that can contribute to the satisfaction of the RR is defined as follows:

$$S_{RR} \coloneqq \{X_{e,a,d,t} \in \mathcal{X}_{RR} | a = RR.activity, d = RR.date, t = RR.time\}$$

Demand constraint can be formulated as follows:

$$\sum_{X_{e,a,d,t} \in S_{RR}} X_{e,a,d,t} = RR.count \ \forall RR,$$

This is the basic formulation that is sufficient in case we have enough employees to cover all requirements. In praxis, however, it is not always the case. The task to find out, whether there are sufficient resources available to meet all the requirements, is itself hard. One possibility is to turn our model to solver and see, if there is a feasible solution. This is the last resort solution, as we don't want to waste computation time. Instead we will run a series of tests to identify the inconsistences in requirements without proceeding any further.

Pre-processing Resource Requirements

Before we run the optimization, we will perform some pre-processing of the set of rostering requirements and statically check whether we dispose of large enough number of employees and if the composition of skills the employees have is

sufficient. All of these conditions have to hold for the problem to have a feasible solution. The manager of the contact center may use these tests when preparing data to formulate the requirements for resources in a consistent way. However, there are cases in which these condition hold but a feasible solution may not exist anyways. The benefit of the pre-processing is to filter out at least some inconsistent input. We have used these pre-processing scenarios:

Single Resource Requirement Check

For each resource requirement we count the number of employees eligible for being scheduled based on the current requirement.

$$\sum_{X_{e,a,d,t} \in S_{RR}} X_{e,a,d,t} \geq RR.count \quad \forall RR,$$

This is actually a static check of the condition prior to model creation which has to hold later in the model. The possible obstacles for scheduling an employee are:

- employee cannot work the required activity
- employee cannot work during the time requested
- employee has an approved day off

If there are at least RR.count employees available, this check is passed. If there is shortage of staff, we can let the manager know about problem causing requirements together with the resource gap.

Per Activity Daily Summation

First we count the demand for each activity and shift on the current day. Then we compare it to the total number of employees capable of working that activity on the particular shift that does not have a day-off. We do this for each activity on each day and activity.

$$\sum_{\substack{e \in E, t \in [0..24] \\ X \in \mathcal{X}_{RR}}} X_{e,a,d,t} \geq \sum_{\substack{RR.date=d, \\ RR.activity=a}} RR.count \quad \forall d \in plan.period, \forall a \in A$$

Overall Daily Summation

For each day we will count the total number of required employees. We will compare this number to the total number of agents available on that particular day (basically check if the employee doesn't have a day-off).

$$\sum_{\substack{e\in E, a\in A, t\in[0..24] \\ X\in \mathcal{X}_{RR}}} X_{e,a,d,t} \geq \sum_{RR.date=d} RR.count \quad \forall d \in planning\ period,$$

This check can fail even though both of the previous checks hold. Here we check the necessary condition on staffing levels for a single day. Still, it is not a sufficient condition.

Introducing Dummy Agents

It may be the case that the manager of a call center needs to specify requirements that cannot be satisfied with available resources. Whether the manager knew it already or was informed about it by the pre-processing tool, he might want to be able to generate the roster anyway. The reason is that the shortage of staff can be covered by hiring agency workers for precisely those shifts, where there is lack of in-house resources.

To be able to generate a roster, we need to alter the demand constraint so that it will hold even with low number of employees. After obtaining results from the solver we will also need to know which RR couldn't be satisfied because of the understaffing. This leads us to introducing a set of dummy variables - *dummy agents*. We want this set to be possibly small. Therefore we have only one dummy agent for each activity, shift and day (such agent can 'work' arbitrary number of shifts in parallel).

We can now reformulate the constraint as

$$\sum_{X_{e,a,d,t}\in S_{RR}} X_{e,a,d,t} + X'_{a,d,t} = RR.count \quad \forall RR,$$

$$X'_{a,d,t} \in [0, +\infty)$$

We also want to penalize the use of these dummy agents, because they can only be used when there is no regular agent at disposal. Therefore the objective function

coefficient for this variable $c_{X'_{a,d,t}} := |E|^2 \times UnderstaffingFactor$. This prohibitive rate is so great that there is no meaningful tradeoff that can outweigh the unnecessary use of *dummy agents*.

3.5.2. Optional Demand

In case of ORRs, there is no requirement on setting any variables to true. However, there is an upper bound on the number of roster items generated from an ORR. The set of variables that correspond to the *ORR* is defined as follows:

$$S_{ORR} := \{X_{e,a,d,t} \in \mathcal{X}_{ORR} | a = ORR.activity, d = ORR.date, t = ORR.time\}$$

This leads to the following constraint:

$$\sum_{X_{e,a,d,t} \in S_{ORR}} X_{e,a,d,t} \leq ORR.\max_count \ \forall ORR,$$

As we have mentioned, ORRs should be satisfied according to the priority. Hence the objective function coefficient for this variable is going to be altered as follows:

$$c_{X_{e,a,d,t}} := c_{X_{e,a,d,t}} - ORR.priority \times ORR_factor.$$

3.5.3. Inter-shift Breaks

The inter-shift breaks are always connected to the two neighboring roster items of some employee. The variable corresponding to the earlier starting roster item will be called the *leading variable*. We are going to create a constraint instance for each roster item variable present in the model (based on RRs and ORRs). This variable is going to be the leading variable for the current constraint. We have identified two situations in which this constraint has to be implemented differently:

No Shrinking

This first situation occurs when the leading variable has **none** of the '*is from past roster'* and the *'is from input roster'* flag set (it has not been decided yet if the prospective roster item connected to this variable is going to be created or not). This is the case for the majority of variables. We want to forbid the combination of

simultaneously setting the leading variable and any other roster item variable from the set of colliding variables $Y_{X_{e,a,d,t}}$ to true. Firstly we need to identify the set of colliding variables $Y_{X_{e,a,d,t}}$ with the leading variable $X_{e,a,d,t}$.

A variable $Y_{e',a',s',t'} \in Y_{X_{e,a,d,t}}$ if e'=e and the shift *s'* on the day *t'* starts prior to the end of shift *s* on day *t* plus twelve hours (break is shorter than twelve hours). Now if there were any colliding variables for the currently processed *leading variable*, we add a new constraint of following form:

$$X_{e,a,d,t} + \sum_{Y \in Y_{X_{e,a,d,t}}} Y \leq 1$$

$$for\ \forall X_{e,a,d,t} \in \mathcal{X}_{(O)RR}$$

This constraint may not seem to be sound in the following situation: There happen to be two shifts from the input roster or past roster or PR in the $Y_{X_{e,a,d,t}}$(which is actually the setting of the following section). Then there may be a roster item variable in the model with the d and t 'close' or 'equal' to the ones of the earlier IR(PR) variable and thus the inequality would not hold even if we set the leading variable to zero. (0+2 > 1). Such situation however will never occur as we would not generated any roster item variable that would be 'closer' to any IR(PR) variable than twelve hours in the first place.

Shrinking the Breaks

Although we do not exploit the possibility of shrinking the inter-shift breaks actively when creating roster items based on (O)RRs, we allow the manager to use this relaxation in the input roster on manually inserted roster items.

The conditions for this situation to occur are:

- The leading variable is either based on a premade roster item or on a roster item from a past roster (the variable has been explicitly set to true)

AND

- There exists another variable from the input or previous roster whose *end time* plus twelve hours is greater (later) than the *start time* of the leading variable. In other words there is a roster item in the input/past roster preceding the prospective roster item based on the leading variable and the

break between these two is less than twelve hours. We label the set of such variables as *S* (this set will probably contain just one item).

We need to compute the length of break between the shift represented by the leading variable and a preceding one.

$$break\ length \coloneqq \min(X.start - s.end | s \in S, X\ is\ the\ leading\ variable)$$

If the break length is less than twelve hours, we need to extend the time interval for finding variables of the set $Y_{X_{e,a,d,t}}$ by $\min(\min_break_length, break\ length)$. The *minimal_break_length* is according to Czech labor code equal to 4 as breaks can be reduced down to 8 hours at the most. The constraint then has the same form. The difference is in the set of variables in the summation.

$$X_{e,a,d,t} + \sum_{Y \in Y_{X_{e,a,d,t}}} Y \leq 1$$
$$for\ \forall X_{e,a,d,t} \in \mathcal{X}_{(O)RR}$$

3.5.4. Weekly Rest

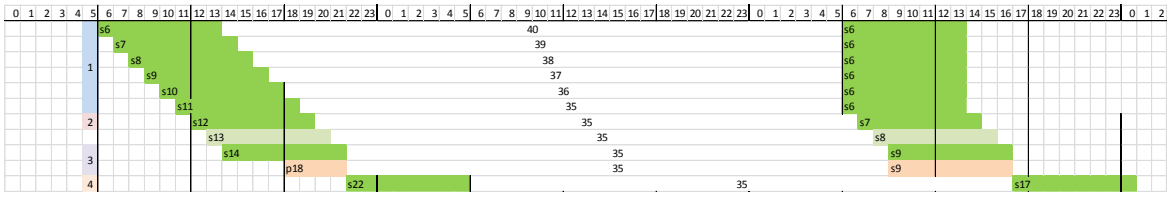
We need to guarantee for each employee during any given period of seven consecutive days that within this very timespan there exists a break of minimal length of thirty five hours. Firstly we need to specify the breaks.

Example: We say that there is a break on day d of type t1 for employee e if there are no roster items overlapping with a time period starting on day d-1 at 19:00 and ending on day d+1 at 6:00.

We chose the time step of one hour. Then there are 24 possibilities of daytime when a break can start. The number of reasonable break types depends on the distribution of start times of the (O)RRs. If there happen to be some 'starting times' missing (e.g. there is no requirement starting at 5:00AM) we may merge some break types.

It would be sufficient to consider just one break type to meet the requirement of the weekly rest. However, there might be rosters that also meet the weekly rest requirement, but with some other break composition (using other break types) gaining a better objective value for instance. Therefore we want to group as many categories of breaks without cutting the solution space.

In the case of the problem instance solved by this paper we have identified four categories that contain all possible break types.



In this figure we list all possible starting times of the last shift before the break. Then we find the earliest shift that can follow so that there is a thirty five hours break between the end of the first shift and the start of the following one. Because there are no shifts starting between 22:00 and 6:00 we can group the first six types into one. The set of break types is denoted as *BT*.

Break Helper Variable

In the example of a break type above we have mentioned the *day* parameter. Given the *break type* and *day* we can specify the precise time interval of the break. For each break instance we create a helper variable $T_{e,d}^{type}$. The set of roster item variables incident to break instance represented by a helper variable $T_{e,d}^{type}$ is defined as follows:

$$S_{T_{e,d}^{type}}:$$
$$= \{X_{e,a,d,t} \in \mathcal{X}_{(O)RR} \big| [X_{e,a,d,t}.start, X_{e,a,d,t}.end] \cap [T_{e,d}^{type}.start, T_{e,d}^{type}.end] \neq \emptyset\}$$

We bind the helper variable $T_{e,d}^{type}$ with the corresponding roster item variables.

$$\sum_{X_{e,a,d,t} \in S_{T_{e,d}^{type}}} X_{e,a,d,t} - bigC(1 - T_{e,d}^{t}) \leq 0$$

$$for\ \forall e \in E, \forall type \in break\ types, \forall d \in pl\ period$$

The *bigC* is a non-negative big integer that is greater than the number of shifts that can be assigned to one employee during thirty five hours. If the *bigC* was too small, than the helper variable would not work properly. In practice this constant can be set to 3 or 4. For defensive approach we can set it to any large number like 100.

The normalized formulation is

$$\sum_{X_{e,a,d,t} \in S_{T_{e,d}^{type}}} X_{e,a,d,t} + bigC * T_{e,d}^{t} \leq bigC$$

$$for\ \forall e \in E, \forall type \in break\ types, \forall d \in pl\ period$$

This constraint entails following quality to the roster:

$$T_{e,d}^{t} = 1 \Rightarrow\ there\ is\ a\ break\ 'around'\ day\ \boldsymbol{d}\ of\ type\ \boldsymbol{t}\ for\ employee\ \boldsymbol{e}.$$

We exploit this fact to conclude the constraint that will enforce the weekly rest roster feature.

$$\sum_{d=d_0}^{d_0+7} \sum_{t \in BT} T_{e,d}^{t} \geq 1$$

$$for\ \forall d_0 \in [pl\ period.start.date, pl\ period.end - 7],$$
$$\forall e \in E, \forall d_0 \in (pl\ period \setminus last\ 6\ days)$$

3.5.5. Employee Requests: Time off Requests

Part of the input is a list of time off requests. We will be processing requests with status set to *Auto* and *Approved* differently. We start with finding the set of variables that overlap with the currently processed request *r*. Based on the value of the *CWT Component* attribute the set differs: If ***CWT Component*** **is true**, we select all variables that overlap with any day that overlaps with the request. Hence, it is not necessary for the variable to directly overlap with the request to be selected to the set:

$$S_r \coloneqq \{X_{e,a,d,t} \in \mathcal{X}_{(O)RR} | d = r.day\}$$

If ***CWT Component*** **is false** we select only variables directly overlapping with the request:

$$S_r \coloneqq \{X_{e,a,d,t} \in \mathcal{X}_{(O)RR} | [X_{e,a,d,t}.start, X_{e,a,d,t}.end] \cap [r.start, r.end] \neq \emptyset\}$$

Approved Requests

If the request is approved, we need to make sure, that there will be no shifts scheduled during this request. We achieve this by setting all variables from the S_r set to false (to zero).

$$X_{e,a,d,t} := 0$$
$$for\ \forall X_{e,a,d,t} \in S_r$$

Auto Requests

If the status is set to *auto*, it is up to system to decide if it is possible to satisfy this request. The likelihood of success depends on the seniority of the employee, on the priority and mostly on the available resources. These attributes combined will be added to the objective function coefficient of variables involved.

There is a simple workflow determining the order of satisfaction of a time off request:

No activity scheduled > activity not counting to CWT > activity counting to CWT

The motivation for this ordering is that if an employee asked for time off, it is better to get a shift like emergency (non CWT activity) and remain off-site in majority of cases than being scheduled to work on-site on some activity that counts to CWT. For that purpose there is the *TORCWT_multiplier* constant. The domain of this multiplier is $\boldsymbol{R} \cap [\mathbf{0}, \mathbf{1}]$. It is up to business analysis to determine the most suitable value of this parameter. In case of activities counting to CWT, the multiplier is equal to 1.

The objective function coefficient gets increased by following value *time off coefficient* in case of variable

$$c_{X_{e,a,d,t}} := c_{X_{e,a,d,t}} + TOR_{coeff} \times TORCWT_{multiplier}$$
$$for\ \forall X_{e,a,d,t} \in S_r$$

TOR_coeff is subject to parameterization, often set equal to 1. We may also add the seniority of employees to the formula as well as other coefficients based on business knowledge.

3.5.6. Contract Working Time Enforcement

The key issue in scheduling workforce is to ensure that the contracts for individual employees will never be broken. There are various contracts present in the contact center, each determining maximal number of shifts the employee is supposed to work during the planning period N_e, where *e* is the employee index.

There were just eight hour shifts counting to contract working time in case of the customer. Therefore it is acceptable to enforce the contracted working hours based on the number of shifts, not considering the number of hours each employee has to work. This approach has actually been explicitly required by the customer. It is possible to implement a hour-based version of this constraint following the same ideas.

We need to design a constraint that will enforce the employee's contract for each employee. The idea is to simply require the number of shifts worked by an employee to be equal to the number of shifts the employee is required to work. It leads to the following constraint:

$$\sum_{a,d,t} X_{e,a,d,t} = N_e$$

The problem of this formulation is that it doesn't take the days-off into account.

Example: John Smith works halftime. Imagine he happens to be ill or on a planned holiday that has been correctly approved ahead for two weeks. In such case system should schedule working shifts for him only during the rest of the time. If the planning period is one month, the remaining approximately two weeks would remain for scheduling Mr. Smith to shifts. The total working time restriction in form mentioned above would leave room for scheduling work for Mr. Smith as a full time employee during those two weeks, which is clearly a mistake.

It shows that we need to subtract the number of days-off that the employee was granted (but only those that count to CWT – like holiday, but not like simple time off). This can be calculated from all the employee's approved time off requests. We will denote it $N_{TO_{approved}}$.

$$\sum_{a,s,t} X_{e,a,s,t} \leq N_e - N_{TOR_{approved}}$$

The most complicated are the time off requests whose status is set to *auto* (it is left for system to decide whether they will come true or not). By the time of constructing the constraint we don't know which value is going to be assigned to which variables. As a matter of fact we don't know whether we are supposed to subtract corresponding number of shifts or not.

However, we can come up with a formulation solving this problem too. We want to subtract one day from the N_e for each *auto time off* request that is going to be fulfilled. When a time off request is fulfilled, all decision variables for that particular employee, and time interval will be equal zero (false). On the contrary, if such request is denied, there will be exactly one roster item variable of the set mentioned assigned to true and the rest will remain assigned zero[3].

If

$S_{TOR} \coloneqq \{X_{e,a,d,t} \in \mathcal{X}_{(O)RR} | [X_{e,a,d,t}.st., X_{e,a,d,t}.end] \cap [TOR.st., TOR.end] \neq \emptyset, \}$

is the set of variables incident with the time off request *TOR*,

$$1 - \sum_{X_{e,a,d,t} \in S_{TOR}} X_{e,a,d,t} = \begin{cases} 0 \; if \; request \; is \; denied \\ 1 \; if \; request \; is \; approved' \end{cases}$$

Now we can embed the *auto* time off requests into the constraint

$$\sum_{a,d,t} X_{e,a,d,t} \leq N_e - N_{TOR_{approved}} - \sum_{TOR \in TOR_{Auto}} \left[1 - \sum_{X_{e,a,d,t} \in S_{TOR}} X_{e,a,d,t} \right] \forall e \; E$$

We conclude with the standardized form of the constraint

$$0 \leq \sum_{a,d,t} X_{e,a,d,t} + \sum_{TOR \in TOR_{Auto}} \sum_{X_{e,a,d,t} \in S_{TOR}} X_{e,a,d,t} \leq N_e - N_{TOR_{approved}} - N_{TOR_{Auto}}$$
$$for \; \forall e \in E$$

[3] This follows from the fact that we deal with eight hour shifts and the shift distribution disallows two roster item variables to be set to true on just one day

3.5.7. Employee Requests: Want-to-Work Requests

The *Want-to-Work* requests are a counter notion to the *Time off* requests. An employee wants to work during a specified time period. We will proceed similarly to the case of *Time off* requests. We start with the identification of the set of variables overlapping with the currently processed request $r \in WTWreqs$.

$$S_r := \{X_{e,a,d,t} \in \mathcal{X}_{(O)RR} | X_{e,a,d,t}.start \cap [r.firstStart, r.lastStart] \neq \emptyset\}$$

Approved Requests

If the actual request is approved, we need to make sure, that the employee will be assigned a shift starting at time specified. Therefore we add a following constraint:

$$\sum_{X_{e,a,d,t} \in S_r} X_{e,a,d,t} = 1$$
$$for\ \forall WTWreqs$$

Approving a *Want-to-Work* request can possibly lead to an infeasible model. To be able to generate a roster nevertheless the impossibility of satisfying the request we introduce a new class of dummy variables. The dummy variable $Xw'_{e,d,t}$ is a binary variable with a large objective function prohibitive rate $WTWviolationFactor$. The constraint with dummy variable is as follows:

$$\sum_{X_{e,a,d,t} \in S_r} X_{e,a,d,t} + Xw'_{e,d,t} = 1$$
$$for\ \forall WTW\ request\ \boldsymbol{r}$$

The approved want-to-work requests and roster items from the premade roster are very similar. The only difference seems to be the fact that in case of approved WTW requests we do not specify the activity. It is important to note, that another critical difference is that the approved WTW request doesn't offer the break-shrinking feature as the premade roster items do. Hence, the sequence of a roster item from premade/past roster followed by an approved WTW request with the break between the two shorter than twelve hours causes the problem to be infeasible.

Auto Requests

If the status is set to *auto*, it is up to system to decide if it is possible to satisfy this request. The likelihood of success depends on the seniority of the employee, on the priority and mostly on the available resources. These attributes combined will be added to the objective function coefficient of variables involved.

The objective function coefficient gets decreased by following value *want-to-work coefficient* in case of variable

$$\forall X_{e,a,d,t} \in S_r : c_{X_{e,a,d,t}} := c_{X_{e,a,d,t}} - WTW_coeff$$
$$for\ \forall TOR$$

WTW_coeff is subject to parameterization, often set equal to 1. We may also add the seniority of employees to the formula as well as other coefficients based on business knowledge.

3.5.8. Employee-Activity Limits

To restrict the number of shifts an employee is going to be scheduled on some specified activity, for each $EA_lim_{e,a}$ the set of variables involved is:

$$EA_set_{e',a'} := \left\{ X_{e,a,d,t} \in \mathcal{X}_{(O)RR} \,\middle|\, e' = e \;\&\; a' = a \right\}$$

The constraint enforcing the limits looks as follows:

$$EA_lim_{e,a}.min \leq \sum_{X_{e,a,d,t} \in EA_{set_{e',a'}}} X_{e,a,d,t} \leq EA_lim_{e,a}.max, \forall EA_lim_{e,t}$$
$$for\ \forall EA_lim_{e,a} \in Empl.-Activity\ Limits$$

This constraint is again a possible source of infeasibility that can be caused by incautious manipulation with the limits. Therefore it is possible to add a dummy variable to the constraint. In fact, we need to add a special dummy variable for the violation of the upper and the lower limit too. If there was just one, the domain

would have to be **Z** to make up for both overflow and underflow. It would, however, be impossible to set up restrictive rate as we would make the problem unbounded. $\overline{EA'_{e,a}}$ and $\underline{EA'_{e,a}}$ are the dummy variables allowing for violation the inequalities. The domain is $\boldsymbol{Z}_0^+$ and the objective function coefficient is $c_{\overline{EA'_{e,a}}} = c_{\underline{EA'_{e,a}}} = 50 * |E|$.

The constraint with the dummy variables is formulated as:

$$EA_lim_{e,a}.min \leq +\underline{EA'_{e,a}} + \sum_{X_{e,a,d,t} \in EA_set_{e',a'}} X_{e,a,d,t} - \overline{EA'_{e,a}}$$
$$\leq EA_lim_{e,a}.max, \forall EA_lim_{e,t}$$
$$for\ \forall EA_lim_{e,a} \in Empl. - Activity\ Limits$$

3.5.9. Employee-Shift Limits (*ES_lim*)

The implementation of the employee-shift limits goes along the same lines as the one of employee-activity limits. For each $ES_lim_{e,t}$ the set of variables involved is:

$$ES_set_{e',t'} := \{X_{e,a,d,t} \in \mathcal{X}_{(O)RR} \mid e' = e \ \& X_{e,a,d,t}.start.hour \cap [t, t+1] \neq \emptyset\}$$

The actual constraint is formed as:

$$EA_lim_{e,t}.min \leq +\underline{ES'_{e,t}} + \sum_{X_{e,a,d,t} \in ES_set_{e',t'}} X_{e,a,d,t} - \overline{ES'_{e,t}}$$
$$\leq ES_lim_{e,t}.\max, \forall EA_lim_{e,t}$$
$$for\ \forall ES_lim_{e,t} \in Empl. - Activity\ Limits$$

The same holds for the dummy variables as in case of *EA limit* dummy variables.

3.5.10. Employee-Activity Priority

The implementation of this feature is straight-forward. When we generate any roster item variable, we check the employee-activity priority list and alter the objective function coefficient respectively:

$$c_{X_{e,a,d,t}} := c_{X_{e,a,d,t}} - EAP_{multiplier} * EAP[e,a]$$
$$for\ \forall X_{e,a,d,t} \in \mathcal{X}_{(O)RR}$$

The $EAP[e,a]$ is the priority coefficient for employee *e* on activity *a*. The $EAP_{multiplier}$ is subject to parameterization, often set to 0.01.

3.5.11.Employee Time Preference (*ETP*)

The employee time preference is technically very close to the *want-to-work* request with the status set to *auto*. The main difference is that the *ETP* is applied to roster item variables during whole planning period and the value added/subtracted from the objective function value are different (usually smaller). The set of variables corresponding to the employee *e* and time of start between *t* and *t+1* (again we choose the one-hour time step) is:

$$ETPvars_{e,t} := \{X_{e,a,d,t} \in \mathcal{X}_{(O)RR} | X_{e,a,d,t}.start.hour \cap [t, t+1] \neq \emptyset\}$$

The objective function coefficients are altered as follows:

$$c_{X_{e,a,d,t}} := c_{X_{e,a,d,t}} - ETP[e,t].priority * ETP_{multiplier}$$
$$for\ \forall X_{e,a,d,t} \in ETPvars_{e,t}$$

3.5.12.Fair Policy - Activity

We want that all employees have balanced rosters. The activity distribution should be equal for all employees. The idea is to compute the expected number of shifts of a given activity and then try to schedule all shifts in such fashion that these expected numbers will align with the actual ones.

Expected Number of Shifts

For each activity we need to state the appropriate number of shifts that the employee should work. The total number of required man-days needed for an activity *a* is denoted $RR[a]$. Next figure we need is the number of opportunities for an agent to work the particular activity. We denote that number as $Slots(e,a)$ and compute it as the number of (optional) rostering requirements demanding activity *a*, for which the employee *e* is eligible. Lastly, we need the total number of opportunities (for all agents together) the activity can be worked. We call it $TotalSlots(e)$, but in fact it is the summation of $\sum_{e'} Slots(e',a)$.

The expected number of times an employee *e* should work the activity *a* is:

$$E_{e,a} = |RR[a]| * \frac{Slots(e,a)}{TotalSlots(e)} = |RR[a]| * \frac{Slots(e,a)}{\sum_{e'} Slots(e',a)}$$

Carry Past Results

Carrying past results from last roster cannot be done for the whole roster because shift scheduling is usually carried out in the middle of the planning period of the 'past' roster. Managers have to alter the rosters according to things like sick days of employees. Therefore we cannot use the whole 'past' roster because we don't know what to count for days that haven't passed yet. Instead we will use some portion of the one before last roster.

We will denote the difference between the expected value and the actual one in the past roster as $E'_{e,a}$, $where\ e \in Employee, a \in Activity$. If we choose to carry the past results, we simply add the $E'_{e,a}$ to the expected value for the actual planning period.

Optimizing Per-Agent

Now we have all the pieces we need to carry on with the fair policy implementation. We want the number of scheduled activities for each user to be as close to the expected value.

The first approach is to minimize the difference between expected number of shifts and the actual number of shifts scheduled. As the objective function is minimization, we want to include following expression into the objective function for each compatible and requested combination of employee and activity:

$$|E_{e,a} - \sum_{s,t} X_{e,a,s,t}|$$

The problem is that there is no function for absolute value available. We can only use linear expressions. Therefore we will have to introduce a helper variable $Y_{e,a}$ and two extra constraints of form:

$$E_{e,a} - \sum_{s,t} X_{e,a,s,t} \leq Y_{e,a} \quad (1)$$

$$-E_{e,a} + \sum_{s,t} X_{e,a,s,t} \leq Y_{e,a} \ (2)$$

These two constraints need to be turned into the standard linear constraint form:

$$-Y_{e,a} - \sum_{s,t} X_{e,a,s,t} \leq -E_{e,a} \ (1)$$

$$-Y_{e,a} + \sum_{s,t} X_{e,a,s,t} \leq E_{e,a} \ (2)$$

We will now append the "$+FPcoeff * Y_{e,a}$" expression to the objective function and add both rows to the model.

This approach has, however, one possible drawback. In general the assignment will be quite fair, but even if the total penalty for deviating from the expected value is low, nothing ensures that the deviations are distributed equally. The penalty is the same no matter the distribution.

<u>Example</u>: Employee E_1 has for activity A_1 expected value $E_{1,1}$ = 3. The same holds for all employees E_1 …E_N. For some reasons there is a total positive deviation from the expected value over all agents equal to 10.

Now there has to be the same sum of negative deviation. We will get the same penalty in case that say five employees lack two activity scheduled to meet the expected value as well as in case that just one employee will deviate and in his case the difference 10.

The possible solution would be to include non-linearity into the objective function, say "$+(Y_{e,a})^2$" instead. It would favor the even distribution of slacks.

We would have to linearize this first to be able to include this component into the linear objective function.

Another possible way would be to minimize the maximal difference between the expected and actual sum of activities. It is straight forward to achieve. Instead of introducing a $Y_{e,a}$ for each employee, we introduce just one helper variable Y_a and use it instead of each occurrence of the $Y_{e,a}$ variable.

The higher the values of priority the more likely are the corresponding variables going to be set to true. The $ETP_{multiplier}$ is subject to parameterization. It can be set to values around 0.1-0.5.

3.5.13. Fair Policy Trade-off

The employee time preference, time off requests, want-to-work requests, activity and shift limits etc. are direct opposite to the concept of fair policy. Fair policy aims to distribute the shifts evenly in terms of starting times and activity composition. If we want to enforce the fair policy, we end up not satisfying many employee time preferences and vice versa. It is practical to perform a deeper business analysis and decide for one of the following concepts:

- Fair policy – suitable in larger teams where the complexity of managing individual preferences is too high
- Employee Preferences + Employee Time/Activity Limits – the possible extreme discrepancies caused by ultimate satisfaction of employee preferences can be reduced by stating explicit limits on some critical shifts or activities. These limits may be learned during time and adjusted as needed.

3.5.14. Time off Variables Introduction

We have described the notion of *time off variables*. To make this concept work we need to bind each time off variable with the set of roster items variables whose time period overlaps with the one of the time off variable. This we achieve by the following constraint.

$$\sum_{\substack{a \in A, \\ t \in [0,24)}} X_{e,a,d,t} - bigC(1 - F_{e,d}) \leq 0 \; \forall d \in pl.period, \forall e \in E$$

The bigC constant has to be greater than the number of $X_{e,a,d,t}$ for the currently processed employee *e* and day *d*. This value can be learned when selecting the variables for the summation. As a result of the constraint, there is either one or no shift scheduled for the current employee during one day. This constraint as well as the notion of time off variables is optional.

3.5.15. Time off Distribution

Grouping Days-off

This feature will be maximizing the number of two days-off in a row. To do so, for each employee *e* and each day (except for the last one) of the planning period *d* we find the set of variables that overlap with the day *d* and the following day.

$$D^2_{e,d}$$
$$\coloneqq \left\{X_{e,a,d,t} \in \mathcal{X}_{(O)RR} \middle| \left[X_{e,a,d,t}.start, X_{e,a,d,t}.end\right] \cap [d.00{:}00, (d+1), 23{:}59]^4 \neq \emptyset\right\}$$

We introduce a helper variable $DoG_{e,d}$ (days-off grouping variable) for each such two days and an employee.

$$\sum_{X_{e,a,d,t} \in D^2_{e,d}} X_{e,a,d,t} - bigC * DoG_{e,d} \leq 0$$
$$for\ \forall e \in E, d \in pl\ period$$

The *bigC* constant has a similar meaning as in the case of *break helper variable*. It is a non-negative big integer that is greater than the number of variables in the $D^2_{e,d}$ set that can be simultaneously set to true. In practice this constant can be set to 3 or 4. For defensive approach we can set it to any large number like 100.

The constraint entails following: if the $DoG_{e,d}$ is equal zero, there is no shift scheduled to the employee *e* on the day *d* and *d+1*. We exploit this fact by setting a negative coefficient to the $DoG_{e,d}$ variable in the objective function:

$$c_{DoG_{e,d}} \coloneqq -daysOff_{groupingCoefficient}$$

As the objective function is minimization, the number of couples of days-off is going to be maximized. The $daysOff_grouping_coefficient \in R^+$ is subject to parameterization. It can be set to 0.1 for instance. This strategy could be done to higher degree. We could group triples and longer series of days-off. The marginal

[4] The notation *d.00:00* means starting midnight of the day *d*. Similarly *d.23:59* means end of the day *d*.

utility drops with each higher degree as the grouping effect barely increases while the complexity of the model grows.

Penalizing Single Day-off

Another possibility of achieving suitable distribution of days-off is to penalize the pattern *working day –day off – working day*. Let us find the formula that will determine whether there is this pattern present in given three consecutive days. We use the *time off variables* and will analyze all combinations of shifts. We need an expression whose value determines whether there is the pattern we want to penalize. Recall the concept of the binary *time off variables* $F_{e,d}$ that are equal to one if the employee *e* has no shift scheduled on day *d*, and equal to zero otherwise. We will enumerate all combination of time off variables on three consecutive days and evaluate the candidate formula: $-F_{e,d+1} + F_{e,d} + F_{e,d+2}$.

$F_{e,d}$	$F_{e,d+1}$	$F_{e,d+2}$	$-F_{e,d+1} + F_{e,d} + F_{e,d+2}$
0	0	0	0
0	0	1	1
0	1	0	-1
0	1	1	0
1	0	0	1
1	0	1	2
1	1	0	0
1	1	1	1

The combinations marked with green color are acceptable. The red marked combination corresponds to the pattern we wish to eliminate. The candidate formula has the following useful property. The evaluation of the time off variables corresponding to the undesired pattern is the global minimum if the formula equal to -1. Hence to avoid the pattern we come up with the following constraint for each employee *e* and day *d* (except for the last two) and each helper variable $DoG'_{e,d}$:

$$-F_{e,d+1} + F_{e,d} + F_{e,d+2} + DoG'_{e,d} > -1$$
$$for\ \forall DoG'_{e,d}\ introduced$$

If $DoG'_{e,d} = 0$, the penalized pattern is not present on days *d,d+1,d+2*. Hence we exploit this fact again as in the last paragraph by setting a negative coefficient to the $DoG'_{e,d}$ variable in the objective function:

$$c_{DoG'_{e,d}} := +daysOff_{groupingCoefficient2}$$

As the objective function is minimization, the number of unwanted patterns in the resulting roster is minimized. The constant $daysOff_grouping_coefficient2 \in R^+$ is also subject to parameterization. It can be set to 0.1 for instance.

3.5.16.No Emergency after Six Shifts

This feature is an additional special case of the *Weekly Rest* feature in a way. The nature of the *emergency* activity forces a constraint that will ensure that for any employee any given sequence of seven consecutive days that either doesn't start with six days without the *weekly break* or doesn't end with a day with *emergency* scheduled. We can make use of the set of helper variables introduced in the *Weekly Rest* constraint section.

We create following constraint for each employee *e* for all possible sequences of seven consecutive days (d_0 is the first day of the sequence):

$$\sum_{d=d_0}^{d_0+5} \sum_{t \in helper\ types} T_{e,d}^{t} + \sum_{\substack{t \in [0..24] \\ a \cong emerg.}} (1 - X_{e,a,d,t}) \geq 1$$

The first doubly summation stands for the first condition – either there is a thirsty five hours long break within the first six days. The second part is stands for – or there is no emergency on the last day. Lastly we present the standard form of the constraint:

$$\sum_{d=d_0}^{d_0+5} \sum_{t \in helper\ types} T_{e,d}^{t} - \sum_{\substack{t \in [0..24] \\ a \cong emerg.}} X_{e,a,d,t} \geq 1 + \sum_{\substack{t \in [0..24] \\ a \cong emerg.}} 1$$

$$for\ \forall d_0 \in (pl\ period \setminus last\ 6\ days)$$

3.5.17.No Night Shift after Emergency

This constraint prohibits an emergency shift to be followed by a night shift. For each roster item variable corresponding to emergency we find all roster item variables connected to night shifts on the very next day. Then we allow at most one of these variables to be set to true at once.

$$C_{emergency} \coloneqq \{X_{e,a,d,t} \in \mathcal{X}_{(O)RR} | a\ is\ emergency\}$$

$$X_{e,a,d,t} + \sum_{X'_{e,a,d+1,t}\ is\ a\ night\ shift} X'_{e,a,d+1,t} \leq 1$$

$$for\ \forall X_{e,a,d,t} \in C_{emergency}$$

3.6. Complete Model Recap

After having presented the ideas behind all parts of the MIP, we present the formulation of the mixed integer problem we are going to solve.

Minimize:

$$\sum_{X_{e,a,d,t} \in \mathcal{X}_{RR}} c_{X_{e,a,d,t}} * X_{e,a,d,t}$$

$$+$$

$$\sum_{X_{e,a,d,t} \in \mathcal{X}_{ORR}} (c_{X_{e,a,d,t}} - ORR.priority \times ORRfactor) * X_{e,a,d,t}$$

$$+$$

$$\sum_{a,d,t \ in \ (O)RRs} |E|^2 \times \ UnderstaffingFactor * X'_{a,d,t}$$

$$+$$

$$\sum_{WTW \ request \ \boldsymbol{r}} WTWviolationFactor * Xw'_{e,d,t}$$

$$+$$

$$\sum_{EA'_{e,a} \in \boldsymbol{EA \ Limits}} 50 * |E| (c_{\overline{EA'_{e,a}}} * \overline{EA'_{e,a}} + \ c_{\underline{EA'_{e,a}}} * \underline{EA'_{e,a}})$$

$$+$$

$$\sum_{ES_lim_{e,t} \in \boldsymbol{ES \ Limits}} 50 * |E| (c_{\overline{ES_lim_{e,t}}} * \overline{ES'_{e,t}} + \ c_{\underline{ES_lim_{e,t}}} * \underline{ES'_{e,t}})$$

$$+$$

$$\sum_{\substack{\boldsymbol{e \in E} \\ \boldsymbol{d \in pl \ period}}} -daysOff_{groupingCoefficient} * DoG_{e,d}$$

$$+$$

$$\sum_{\substack{\boldsymbol{e \in E} \\ \boldsymbol{d \in pl \ period}}} +daysOff_{groupingCoefficient2} * DoG'_{e,d}$$

where

$$c_{X_{e,a,d,t}}{}^{5} \coloneqq +TOR_{coeff} \times TORCWT_{multiplier} \ - WTW_{coeff} - EAP_{multiplier} * EAP[e,a] - ETP[e,t].priority * ETP_{multiplier}$$

[5] The presence of time-off / want-to-work request based parts of the expression depend on presence of corresponding request.

Subject to:

Demand Constraint

$$\sum_{X_{e,a,d,t} \in S_{RR}} X_{e,a,d,t} + X'_{a,d,t} = RR.count \ \forall RR,$$

$$where\ S_{RR} := \{X_{e,a,d,t} \in \mathcal{X}_{RR} | a = RR.activity, d = RR.date, t = RR.time\}$$

Optional Demand

$$\sum_{X_{e,a,d,t} \in S_{ORR}} X_{e,a,d,t} \leq ORR.\max_count \ \forall ORR,$$

$$where\ S_{ORR} := \{X_{e,a,d,t} \in \mathcal{X}_{ORR} | a = ORR.activity, d = ORR.date, t = ORR.time\}$$

Time off Variables

$$\sum_{\substack{a \in A, \\ t \in [0,24)}} X_{e,a,d,t} - 100(1 - F_{e,d}) \leq 0 \ \forall d \in pl.period, \forall e \in E$$

Inter-shift Breaks

$$X_{e,a,d,t} + \sum_{Y \in Y_{X_{e,a,d,t}}} Y \leq 1$$

$$for\ \forall X_{e,a,d,t} \in \mathcal{X}_{(O)RR}$$

Where variable $Y_{e',a',s',t'} \in Y_{X_{e,a,d,t}}$ if e'=e and the shift *s'* on the day *t'* starts prior to the end of shift *s* on day *t* plus twelve hours (break is shorter than twelve hours).

Fair Policy

$$-Y_{e,a} - \sum_{s,t} X_{e,a,s,t} \leq -E_{e,a}$$

$$-Y_{e,a} + \sum_{s,t} X_{e,a,s,t} \leq E_{e,a}$$

$$for\ \forall e \in E, \forall a \in A$$

$$where\ E_{e,a} = |RR[a]| * \frac{Slots(e,a)}{TotalSlots(e)} = |RR[a]| * \frac{Slots(e,a)}{\sum_{e'} Slots(e',a)}$$

Weekly Rest

$$\sum_{d=d_0}^{d_0+7} \sum_{t \in BT} T_{e,d}^{t} \geq 1$$

$$for\ \forall d_0 \in [pl\ period.start.date, pl\ period.end - 7],$$

$$\forall e \in E, \forall d_0 \in (pl\ period \setminus last\ 6\ days)$$

We introduce each $T_{e,d}^{t}$ variable with

$$\sum_{X_{e,a,d,t} \in S_{T_{e,d}^{type}}} X_{e,a,d,t} + bigC * T_{e,d}^{t} \leq bigC$$

$$for\ \forall e \in E, \forall type \in break\ types, \forall d \in pl\ period, where$$

$S_{T_{e,d}^{type}}$:

$$= \{X_{e,a,d,t} \in \mathcal{X}_{(O)RR} \big| [X_{e,a,d,t}.start, X_{e,a,d,t}.end] \cap [T_{e,d}^{type}.start, T_{e,d}^{type}.end] \neq \emptyset\}$$

Employee Requests: Time off Requests

$$X_{e,a,d,t} := 0$$

$$for\ \forall X_{e,a,d,t} \in S_r, where$$

$$S_r = \begin{cases} \{X_{e,a,d,t} \in \mathcal{X}_{(O)RR} \big| d = r.day\} \\ if\ r\ is\ CWT\ Component \\ ---- \\ \{X_{e,a,d,t} \in \mathcal{X}_{(O)RR} \big| [X_{e,a,d,t}.start, X_{e,a,d,t}.end] \cap [r.start, r.end] \neq \emptyset\} \\ otherwise \end{cases}$$

Employee Requests: Want-to-Work Requests

$$\sum_{X_{e,a,d,t} \in S_r} X_{e,a,d,t} + Xw'_{e,d,t} = 1$$

$$for\ \forall TOR, where$$

$$S_r := \{X_{e,a,d,t} \in \mathcal{X}_{(O)RR} \big| X_{e,a,d,t}.start \cap [r.firstStart, r.lastStart] \neq \emptyset\}$$

Employee-Activity Limits

$$EA_lim_{e,a}.min \leq \sum_{X_{e,a,d,t} \in EA_{set_{e',a'}}} X_{e,a,d,t} \leq EA_lim_{e,a}.max, \forall EA_lim_{e,t}$$

$$for\ \forall EA_lim_{e,a} \in Empl. - Activity\ Limits, where$$

$$EA_set_{e',a'} := \{X_{e,a,d,t} \in \mathcal{X}_{(O)RR} \mid e' = e\ \&\ a' = a\}$$

Employee-Shift Limits

$$EA_lim_{e,t}.min \leq +\underline{ES'_{e,t}} + \sum_{X_{e,a,d,t} \in ES_set_{e',t'}} X_{e,a,d,t} - \overline{ES'_{e,t}}$$

$$\leq ES_lim_{e,t}.\max, \forall EA_lim_{e,t}$$

$$for\ \forall ES_lim_{e,t} \in Empl. - Activity\ Limits, where$$

$$ES_set_{e',t'} := \{X_{e,a,d,t} \in \mathcal{X}_{(O)RR} \mid e' = e\ \&X_{e,a,d,t}.start.hour \cap [t, t+1] \neq \emptyset\}$$

Grouping Days-off

$$\sum_{X_{e,a,d,t} \in D^2_{e,d}} X_{e,a,d,t} - bigC * DoG_{e,d} \leq 0$$

$$for\ \forall e \in E, d \in pl\ period, where$$

$D^2_{e,d}$

$:= \{X_{e,a,d,t} \in \mathcal{X}_{(O)RR} | [X_{e,a,d,t}.start, X_{e,a,d,t}.end] \cap [d.00{:}00, (d+1), 23{:}59]$[6] $\neq \emptyset\}$

Penalizing Single Day-off

$$-F_{e,d+1} + F_{e,d} + F_{e,d+2} + DoG'_{e,d} > -1$$

$$for\ \forall DoG'_{e,d}\ introduced$$

No Emergency after Six Shifts

$$\sum_{d=d_0}^{d_0+5} \sum_{t \in helper\ types} T^t_{e,d} - \sum_{\substack{t \in [0..24] \\ a \cong emerg.}} X_{e,a,d,t} \geq 1 + \sum_{\substack{t \in [0..24] \\ a \cong emerg.}} 1$$

$$for\ \forall d_0 \in (pl\ period \setminus last\ 6\ days)$$

No Night Shift after Emergency

$$X_{e,a,d,t} + \sum_{X'_{e,a,d+1,t}\ is\ a\ night\ shift} X'_{e,a,d+1,t} \leq 1$$

$$for\ \forall X_{e,a,d,t} \in C_{emergency}, where$$

$$C_{emergency} := \{X_{e,a,d,t} \in \mathcal{X}_{(O)RR} | a\ is\ emergency\}$$

[6] The notation *d.00:00* means starting midnight of the day *d*. Similarly *d.23:59* means end of the day *d*.

Contract Working Time Enforcement

$$0 \leq \sum_{a,d,t} X_{e,a,d,t} + \sum_{TOR \in TOR_{Auto}} \sum_{X_{e,a,d,t} \in S_{TOR}} X_{e,a,d,t} \leq N_e - N_{TOR_{approved}} - N_{TOR_{Auto}}$$

$$for\ \forall e \in E$$

Chapter 4. Implementation

In this chapter we will focus on the practical issues connected to the implementation of the MIP described in the previous chapter. Firstly we need to define what we mean with implementation of MIP. In terms of Chapter 3 we talk about the model as a general set of inequalities with an objective function. In fact, the variables, constraints in form of inequalities and the objective function in form of a linear expression can differ substantially based on the input data. So the implementation of MIP means creating a service that given input data generates the instance of the model. Then of course we must solve the model instance and interpret the results. The interpretation of the results is a roster. The scope of this work is the module capable of parsing the input data, creating MIP model, solving the MIP model and turning solution into a roster. Clearly, we also had to develop a web application that drives the roster creating module to provide the functionality to users. This web application is, however, doesn't belong to main the contribution of the work.

The implementation of the software has been done in Microsoft .NET 4.0 Framework using the C# language.

4.1. Architecture

The whole software solution is formed by two major parts – a *Web Application* and *Calculation Service.* The Web Application is designed to collect and manage data from users as well as present results of computations performed by the calculation service. The calculation service creates, solves and interprets the results of MIP.

The Web Application has been built on the Microsoft Pattern & Practices' Web Client Software Factory 2010 (ver3.0), a guidance implementing the Model-View-Presenter design pattern. The data is stored on Microsoft SQL 2005(8) Server and as the OR mapping we have used the Entity Framework 4.0. The Web Application is deployed on webserver with IIS 7.0. The web application is optimize to work in Internet Explorer as it has been a request from the customer. On the presentation layer we use third-party components by Telerik. These controls offer stable and effective out-of-box functionality.

The computations are time consuming and therefore are performed in the separate service. This service is based on Microsoft Communication Foundation and runs as a separate console application. The service can be hosted on a separate machine for

performance reasons. All computations run in a separate thread and the results are processed asynchronously too. The following figure sketches the architectonical composition of the calculation service as well as a basic flow of computation when generating a roster.

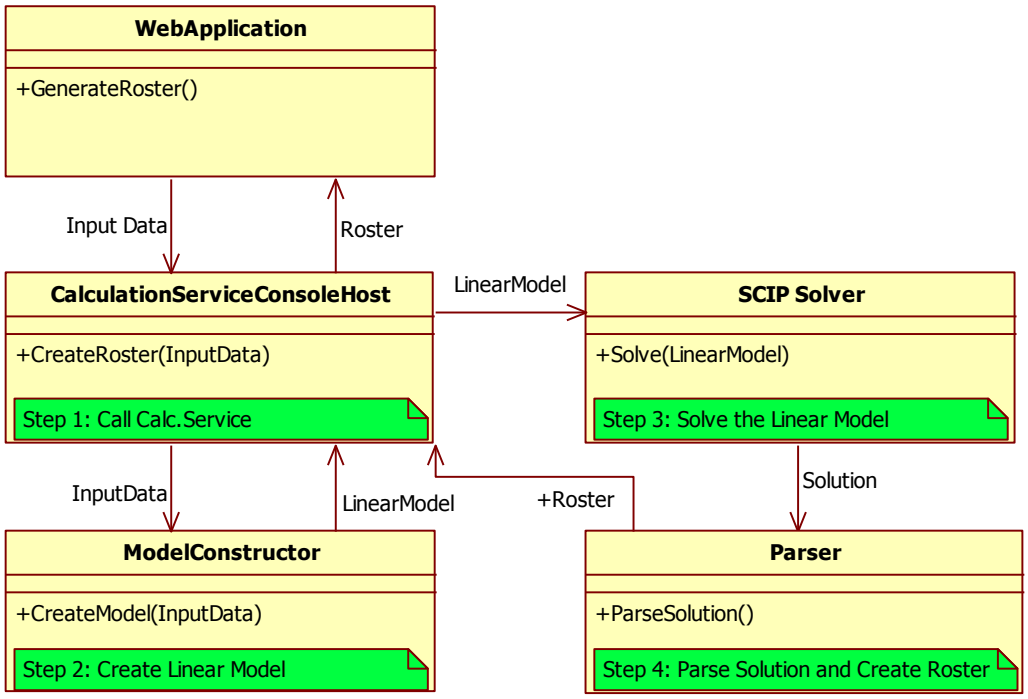


Figure 12: Roster Creation Process

For authorization, authentication, logging and many more we use the Enterprise Library.

Finally, for the MIP representation we use the Microsoft Solver Foundation. This framework offers several approaches to problem solving. We choose the low-level representation of linear model through *ILinearModel* interface where the model is represented as a matrix of coefficients, where rows (constraints) and columns (variables) can be added anytime needed. The objective function is just another row in the model. For indexing variables we use classes that keep track of metadata relevant to each variable. This way we can construct a roster from a set of evaluated variables after acquiring a solution to the model. For solving the MIP we use the SCIP solver (Achterberg, 2007), one of the most efficient non-commercial solvers.

4.2. Testing

We try to cover as much test surface as possible with unit tests. We have performed intensive manual testing on the GUI. We have considered automatic UI testing

(Selenium Framework), however, the extensive use of AJAX makes it very difficult to design use cases for such tool. For testing the soundness of the model we have accumulated number of input data collections that usually test some specific constraint of roster feature. We have found it very practical to design minimalistic test cases as they result in easily readable models (after exporting models to source file). For exporting, we use the *.lp* format as it respects the standard notation of MIP (human readable) and the archaic *.mps* format. The *.mps* format is very useful for debugging purposes together with the *.lp* format as they are complements of each other in a way.

Chapter 5. Justification

In Chapter 2 we have formally specified all features of the rosters we are to generate. We have stated objectives to optimize the rosters. In Chapter 3 we have modeled the problem as Mixed Integer Program. In this chapter we are going to verify whether we have achieved what we have planned to do. The software has been deployed on the customer's site and (by the time of finishing this paper during April 2011) has been used for three month for scheduling workforce. We are going to analyze the input data collected by the application on the site and discuss the problem instance. Then we are going to study the rosters generated on the planning periods of March, April and May. We will analyze the degree to which the rosters accommodate preferences, requests and priorities entered to the system by employees and managers.

5.1. Problem Instance

Both the linear model and its implementation are applicable to various problem instances. In this section we will discuss the application to the real-world instance of the problem. This data come from customer and solving this problem has been the original motivation of this work. Let us analyze the data collected for the purpose of planning shifts for the month of April 2011 for the back-office site.

Employees

There are eighteen employees on the site to be scheduled all of which work fulltime. We have replaced the real names of the employees with random English names. One employee (Lucy) is ill during the whole planning period (April). There is one special class of employees called *supervisors* containing three employees. Alice, Peter, Steve are working as *supervisors*. One of the *supervisors* has to be present on site every day. We will model this as a regular activity and include resource requirement on it.

The managers have introduced following the **limits on activities** for employees. If there is no record in a box for some employee-activity combination, we do not consider any limit. The values are minimal/maximal allowed number of shifts of given activity per planning period.

Emp\Act Limits	Authorization	Oper/Aut	Ticketing	Queue	VIP	Operational	NYC	Night	Emergency	Supervisor	White
Alice									1/2		
Anne	1/-		1/-	1/-					1/3		
Caroline	1/-		1/-	1/-					1/3		
Charlie			1/-	1/-							
Eva	1/-		1/-	1/-							
Hannah	1/-		1/-	1/-					1/3		
Jack	1/-		1/-	1/-					1/3		
Jane	1/-		1/-	1/-					1/3		
Jim	1/-		1/-	1/-					1/3		
Kathy	1/-		1/-	1/-					1/3		
Lucy											
Margaret	1/-		1/-	1/-					1/3		
Michelle	1/-		1/-	1/-					1/3		
Mirka	1/-		1/-	1/-					1/3		
Nadia	1/-		1/-	1/-							
Nicolle	1/-		1/-	1/-					1/3		
Peter									1/2		
Steve											

Figure 13: Employee-Activity Limits

The managers only decided to use the **employee-activity preference** on supervisors. The supervisors do not like to be scheduled to work any other activities than *supervisor* unless the emergency situation requires it. This setting produces acceptable rosters from this perspective.

Employee	Activity	Preference
Alice	Supervizor	0.5
Peter	Supervizor	1
Steve	Supervizor	1

Figure 14: Employee-Activity Priority

The **employee-shift limits** and **employee time preference** (which is in fact employee-shift preference) are used to accommodate wishes of the employees as well as the scheduling strategy of managers. Figure 15 shows both the limits and preferences on various employee-shift combinations. Shifts are annotated as S*i*, *i* being the starting hour of a shift. For each shift-start the minimal and maximal number of shifts per planning period is written in the 'lim' column. The employee's time preference is written in the 'pref' column. As we can see, the critical shifts here are the ones starting between 6 and 8 AM, then afternoon and night shifts. Another interesting fact is that there are no preferences stated for shifts starting at 7.

Especially in case of Jane and Kathy, the preference seems strange. We have found out that many employees commute to work and the preferences are often conditioned by various transportation issues – most of the lines run around 7AM.

The limits and preferences together form the typical soft constraint with an optimization part and a threshold, however, either threshold or optimization part missing is in majority of combinations. Also note that no restriction on minimal number of shifts of any kind is present.

ES pref & limits	S6		S7		S8		S9		S10		S11		S12		S14		S18		S22	
	lim	pref	lim	pref	lim	pref	lim	pref	lim	pref	lim	pref	lim	pref	lim	pref	lim	pref	lim	pref
Eva		-1.0		0.0		0.0		0.0		0.0		0.0		0.0	-/6	0.0		0.0		0.0
Jack	-/6	1.0		0.0		0.0		0.0		0.0		0.0		-1.0	-/6	0.0		0.0	-/2	-1.0
Nicolle	-/6	0.0		0.0		0.0		0.0		0.0		0.0		0.0	-/6	0.0		0.0	-/6	0.5
Anne		1.0		0.0		0.0		0.0		0.0		0.0		0.0	-/4	0.0		0.0	-/2	0.0
Margaret		0.0		0.0		0.0		0.0		0.0		0.0		0.0	-/4	0.0		0.0	-/2	0.0
Alice		0.0		0.0		1.0		0.0		0.0		0.0		0.0		0.0		0.0		0.0
Caroline	-/6	0.0		0.0		0.0		0.0		0.0		0.0		0.0	-/6	0.0		0.0	-/2	0.0
Kathy		-1.0		0.0		-1.0		0.0		0.0		0.0		1.0		0.0		0.0	-/2	0.0
Jim		-1.0		0.0		0.0		0.0		0.0		0.0		0.0		1.0		0.0	-/6	0.5
Peter		-1.0		0.0		1.0		0.0		0.0		-1.0		0.0		0.0		0.0		0.0
Michelle	-/6	0.0		0.0		0.0		0.0		0.0		0.0		0.0	-/6	0.0		0.0	-/2	-1.0
Lucy		0.0		0.0		0.0		0.0		0.0		0.0		0.0	-/6	0.0		0.0		0.0
Charlie	-/6	0.0		0.0		0.0		0.0		0.0		0.0		0.0	-/6	0.0		0.0		0.0
Jane		-1.0		0.0		-1.0		0.0		0.0		0.0		0.0		0.5		0.0		1.0
Nadia		1.0		0.0		0.0		0.0		0.0		0.0		-1.0	-/4	0.0		0.0		0.0
Mirka	-/6	0.0		0.0		0.0		0.0		-1.0		0.0		0.0	-/6	0.0		0.0	-/2	0.0
Hannah	-/6	0.0		0.0		0.0		0.0		0.0		0.0		0.0	-/6	0.0		0.0	-/2	0.0
Steve		0.0		0.0		1.0		0.0		0.0		0.0		1.0		0.0		0.0		0.0

Figure 15: Employee-Shift Limits and Preference

The employee-shift limits and priorities are closely related to the employee-activity kinds as many activities are required to be worked only during certain time. So introducing some limit or preference on some activity may implicitly introduce a limit or preference on some type of shift.

Shifts

Name	S6	S7	S8	S9	S10	S11	S12	S14	Emergency	S22
Duration	6-14	7-15	8-16	9-17	10-18	11-19	12-20	14-22	18-22	22-06

Figure 16: Table of Shifts

In this table we have listed the shifts that can be bound together with activities in the resource requirements. The shifts that employees are allowed to work are shown in the Figure 17.

Employee	S6	S7	S8	S9	S10	S11	S12	S14	S18	S22
Alice	✓	✓	✓	✓	✓	✓	✓	✓	✓	✗
Anne	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Caroline	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Charlie	✓	✓	✓	✓	✓	✓	✓	✓	✗	✗
Eva	✓	✓	✓	✓	✓	✓	✓	✓	✗	✗
Hannah	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Jack	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Jane	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Jim	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Kathy	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Lucy	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Margaret	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Michelle	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Mirka	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Nadia	✓	✓	✓	✓	✓	✓	✓	✓	✗	✗
Nicolle	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Peter	✓	✓	✓	✓	✓	✓	✓	✓	✓	✗
Steve	✓	✓	✓	✓	✓	✓	✓	✓	✗	✗

Figure 17: Allowed Shifts

Most of the employees work all shift, but there are some exceptions. We can see that all but one shift lasts eight hours and the emergency shift only takes four hours. This emergency shift is a special one. It is a shift that, when scheduled, doesn't require the employee to be physically on site. It only requires him to be within the reach to be able to arrive in case of emergency.

Activities

There are eleven activities in total. It is not important what they mean except for the *supervisor* and *emergency* activity. Some activities bind only with a specific shift, some other are scheduled with multiple shifts. Most of the employees are capable of working all the activities except for the *supervisor* activity. In the roster the activities are marked with colors.

Autorizace	Operacní/autorizac	Ticketting	Queue	VIP	Operational
NYC New York	Night	Emergency	Supervizor	Not Specified	

Figure 18: Activities

Planning Period

Planning periods are time intervals for generating rosters which cannot overlap and have to cover the timeline completely. Hence each roster can be assigned to just one planning period. The planning periods here align with one calendar month.

Resource Requirements (RRs)

Resource requirements are the key input. There is a single instance of a RR for each day-time and activity combination desired, however we present a recurrent formulation that is easy to understand.

Activity	Start		End	Mo	Tu	We	Th	Fr	Sa	Su	Count
Autorization	7	-	15	✓	✓	✓	✓	✓	✗	✗	1
Operation/Autorization	11	-	19	✓	✓	✓	✓	✓	✗	✗	1
Ticketting (issue-reissue)	6	-	14	✓	✓	✓	✓	✓	✓	✓	1
Ticketting (issue-reissue)	14	-	22	✓	✓	✓	✓	✓	✓	✓	1
Queue	6	-	14	✓	✓	✓	✓	✓	✓	✓	1
Queue	14	-	22	✓	✓	✓	✓	✓	✓	✓	1
VIP	9	-	17	✓	✓	✓	✓	✓	✗	✗	1
Operation	8	-	16	✓	✓	✓	✓	✓	✗	✗	1
Operation	10	-	18	✗	✗	✗	✗	✗	✗	✓	1
NYC	14	-	22	✓	✓	✓	✓	✓	✗	✗	1
Night	22	-	6	✓	✓	✓	✓	✓	✓	✓	1
Emergency	18	-	22	✓	✓	✓	✓	✓	✓	✓	1
Supervisor	8	-	16	✓	✗	✗	✗	✗	✗	✗	1
Supervisor	9	-	17	✗	✓	✓	✓	✓	✗	✓	1

Figure 19: Resource Requirements[7]

Let's analyze the total demand and offer of resources during the planning period. On the month of April there is demand of 294 man-days of compulsory work. The number of man-days available is 399 (total) – 17 (for approved time-off requests ?? CWT) = 382. This gives us the margin of 88 man-days, which we will use with either ORRs or time-off requests with auto status that count to CWT.

Optional Resource Requirements (ORRs)

Similarly to the case of RRs we present ORRs in a recurrent fashion. All of the requests have the maximal number of employees assigned equal to one. The optionally required activities are *supervisor*, *operation*, *VIP* and *not specified*. The *not specified* activity is actually a well-defined activity from the customer's site during which some specific combination of other activities is done by the employee.

7

Activity	Time			Mo	Tu	We	Th	Fr	Sa	SU	Max	Priority
Supervisor	10	-	18	✓	✓	✓	✓	✓	✗	✗	1	1
Operation	12	-	20	✓	✓	✓	✓	✓	✓	✓	1	1
VIP	12	-	20	✓	✓	✓	✓	✓	✗	✗	1	0.8
Supervisor	12	-	20	✓	✓	✓	✓	✓	✗	✗	1	0.8
Not Specified	10	-	18	✓	✓	✓	✓	✓	✗	✗	1	0.6
Not Specified	8	-	16	✓	✓	✓	✓	✓	✗	✗	1	0.5
Not Specified	9	-	17	✓	✓	✓	✓	✓	✗	✗	1	0.4
Not Specified	11	-	19	✓	✓	✓	✓	✓	✗	✗	1	0.3
Supervisor	9	-	17	✗	✗	✗	✗	✗	✓	✗	1	0.3
Not Specified	12	-	20	✓	✓	✓	✓	✓	✗	✗	1	0.2
Supervisor	10	-	18	✗	✗	✗	✗	✗	✓	✓	1	0.1

Figure 20: Optional Resource Requirements

There is a total number of 211 MDs of optionally requested shifts of activities contributing to the contract working time for the month of April. We can see that there are plenty of ORRs to cover the contracted number of MDs that the employees have to work (88 MDs).

Time off Requests (TORs)

Here we present the overview of time-off requests handed in by employees for the month of April. The majority of requests are the *time off*, then the *illness*, which actually come from just one employee (Lucy). The column paid means if the realized time off counts as a working day. The column *Limited Number* stands for the fact that there can be a limit on the number of requests in some category. Last column stands for total number of requests of the given category handed in by employees.

TOR type	Meaning	Paid	Limited Number	Ttl Number
Time Off	Employee doesn't want to be scheduled.	No	No	108
Holiday	Regular holiday	Yes	Yes	16
Unpaid Time Off	From the application point of view we do not distinguish between this and time off.	No	No	0
Compensatory Time Off	After having worked overtime one gets compensatory time off.	Yes	Yes	1
Illness	Regular illness.	Yes	No	21
Cannot Work	Used for blocking off part of day (agency workers, students..).	No	No	0

Figure 21:Time Off Requests Overview

The following figure contains numbers of requests handed in by individual employees, ratio of approved requests to those let for system to decide as well as total numbers of requests aggregated by status.

	Auto	Approved	Total
Lucy	0	21	21
Charlie	2	12	14
Jane	1	10	11
Jack	9	0	9
Peter	7	2	9
Anne	4	4	8
Mirka	8	0	8
Caroline	5	3	8
Steve	8	0	8
Margaret	3	5	8
Alice	5	3	8
Eva	0	8	8
Nadia	1	7	8
Michelle	3	4	7
Hannah	0	3	3
Kathy	3	0	3
Nicolle	0	3	3
Jim	2	0	2
Margaret	0	0	0
Total	**61**	**85**	**146**

Figure 22: Number of Requests for Employees

Want-to-Work Requests

The numbers of WTW requests grouped by status follows. One request was rejected as there were two requests on a single shift.

Status	Approved	Rejected	Auto
Number	12	2	136

Figure 23: WTW Requests by Status

Another interesting viewpoint is the number of requests handed in by individual employees. Some employees try to schedule almost all the shifts they are obliged to work during the planning period. This is the case of Caroline (21/21), Anne (20/21), Charlie (15/16), and some more. Other employees do not benefit from this feature as much like Steve or Eva. Note the numbers for Mirka – she has total number of 8 WTW requests, from which 2 are rejected (this fact can be observed in Figure 26).

Name	TTL Reqs	Approved	Auto
Anne		0	21
Caroline		4	17
Charlie		0	16
Jane		0	16
Jack		0	14
Margaret		0	14
Nadia		4	6
Alice		0	8
Mirka		0	6
Jim		0	7
Peter		0	6
Michelle		1	3
Nicolle		3	0
Eva		0	1
Steve		0	1

Figure 24: Want to Work Requests Overview

Last interesting report on WTW requests is the time distribution of requests. Again we can see that the critical shifts are the S6 and S14 with S22.

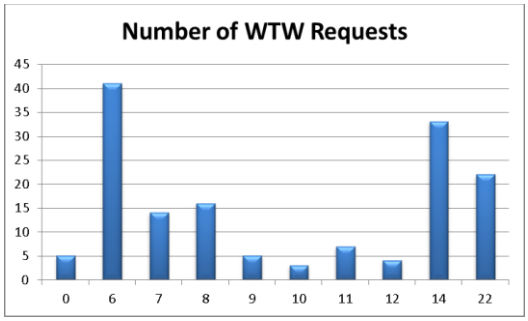


Figure 25: WTW Request Time Distribution

We conclude the analysis of employees' request by a complete overview of requests. The abbreviation *TO* stands for 'Time Off ' type of time off request. The S*i* stands for want-to-work request on shift starting at time *i*. The green color means status *approved*, red color status *rejected* and yellow color status *auto*.

April '11	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30
Steve		V	S12						V	V						V	V						V	V					V
Peter	S8	V	V					S8	V						S8	V					S7	Time Off			S8			S7	V	V
Alice	S8	TO			S11				TO			S11			NV	TO (Time Off)		S11					S9	S8		S11			S9	V
Jack	S7	V	V	S14	S11			S7	V	V	S14			S6	V	S6	S6	V	S11		S7	S6	V	V	S11		S10		S6	V
Mirka		V	V	S14				S6	V	V				V		S6	S6		V			S7	V	V	S14			S22	S22	
Anne	V	V	V	S8	S6	S22	S22		S6	S14	S8	S7	S6	S7	V	V	S14	S6	S8	S6	V	S14	S6	S14	S8	S8	S6	V	V	S6
Hannah	V	V	V																											
Caroline	S14	S14	S14	S22	S22		V	S6	S6	S6	S7	S6	V	V	S6	S14	S14	S8	S6	S9	V	V	S14	S14	S9	S6	S8	V	V	V
Kathy																						V	V	V						
Michelle	V										S6	V	V	V	V	S12									S			S6	V	V
Nadia		TO		S8		S7		S6	Time Off		S8		S7	S6		S6	V	S8		S7		S6	TO (Time Off)							
Charlie	S10	S10	S12	V	S14	S14	S14	S14	S14	S12	V	S14	S14	S14	V	V	V	D	D	D	D	D	V	V	V	S	S	S	V	S
Margaret	S6	V	V	S6	S6			S6	Time Off		S6	S6		S22	S22		S9	S6	S6			S6	Time Off		V	S6			S6	V
Jim		V	V						S14	S14						S14	S14						S22	S22						S14
Eva					S7	Holiday			Time Off		Holiday		D																	
Jane	V	S22	S22		S7	S14	S14	S14	Time Off		Holiday					Time Off		S22	S22	S22	S22	S22			S22	S22	S22	S22	S22	V6
Nicolle		V							V							S14							S14					S6	V
Lucy	Ill			Illness							Illness							Illness							Illness					

Figure 26: Roster Request Overview

Past Rosters

Past roster mainly determines the composition of shifts during the first week on the roster. We show the last seven days fragment of the past roster that has influence on the inter-shift and weekly breaks.

Employee	25.3.	26.3.	27.3.	28.3.	29.3.	30.3.	31.3.
Steve	S8			S12	S8	S9	S8
Peter	S10			S8	S10	S8	S9
Alice	S9		S8	S10	S9	S12	S10
Caroline	S9	S6	S6		E18	S7	S8
Kathy	S14	S14	E18	S14	S14	E18	S7
Jim	S14	S22	S22		S14		
Eva	S11			S9	S14	S14	S12
Jack	S6			S14	S11	S9	S6
Nicolle	S8	S10	S6	E18	S7	S11	S9
Anne	E18	S6	S14	S6		D	D
Margaret	S8				S6	S14	S14
Michelle	S22		S12	S11	S8	S6	E18
Lucy	N			N	N	N	N
Charlie	S10	S8	S8	S6		S8	S6
Jane				S22	S22	S22	S22
Nadia	S6			S8	S9	S6	S14
Mirka	S7	E18	S14	S14	S12	S12	S14
Hannah	S14	S14		S7	S6	S14	S11

Figure 27: Relevant Fragment of Past Roster

Premade Roster

In the month of April there were no premade roster items inserted. However, we have tested this feature on number of artificial premade rosters.

5.2. Computation

Computational experiments have been conducted on several different machines. The solving process doesn't benefit from parallelization and therefore the computation times depend mainly on the processor speed. There has been observed a linear acceleration of computation based on CPU speed. On the computer that was used for generating the April roster, Intel Core 2 Duo U9400 @ 1.40GHz, 4 GB RAM, with 64-bit Win 7 operating system, with the 64bit version of SCIP 1.2.0 the computation took 51 minutes to reach the optimal solution. The first feasible integer solution was found in 6:50 minutes with the gap between primal and dual bound of 4.0 which was 3.00% of the primal bound.

The model for month of April has a total number of 7160 variables (7007 binary, 85 integer and 68 continuous) and 8777 constraints.

5.3. Results

In this section we will show how much the resulting roster fulfills all requirements from section 5.1. We start with the roster that was generated for the month of April. The roster is composed from pairs of rows. We see an example in the following figure.

Employee\Date Roster Requests for the given employee	1.4.	2.4.	3.4.	4.4.	5.4.	6.4.	7.4.	8.4.	9.4.	10.4.	11.4.	12.4.	13.4.	14.4.	15.4.	16.4.	17.4.	18.4.	19.4.	20.4.	21.4.	22.4.	23.4.	24.4.	25.4.	26.4.	27.4.	28.4.	29.4.	30.4.
Anne				S8	S6	S22	S22			S12	S8	S7	S6	S7	S7		S14	S11	S8	S6		S8	S6	S6	S8	S8	S6			S6
Roster Requests	V	V	V	S8	S6	S22	S22		S6	S14	S8	S7	S6	S7	V	V	S14	S6	S8	S6	V	S14	S6	S14	S8	S8	S6	V	V	S6

Figure 28: One Row of a Roster

The upper row starting with the name of employee contains roster items for the employee. The columns represent days in the planning period. The roster items can be of several types – **S*i*** (regular working shift starting at *i* hours; the color represents the activity – see the table of resource requirements), **N** (illness), **D** (holiday), **NV** (compensatory time off), and an empty that stands for emergency (S18). All the other cases (empty cell) stand for time off.

The lower row contains roster requests for the same employee. The requests are either want-to-work requests of the type **S** (any shift without closer time specification), **S*i*** (any shift without closer time specification) ***Vi*** (time off starting at specified time), **V** (daylong time off). The color of the requests doesn't mean activity but status. Green requests are *approved* and the orange ones are set to *auto*. Comparing both rows we can easily tell which requests were fulfilled and which not.

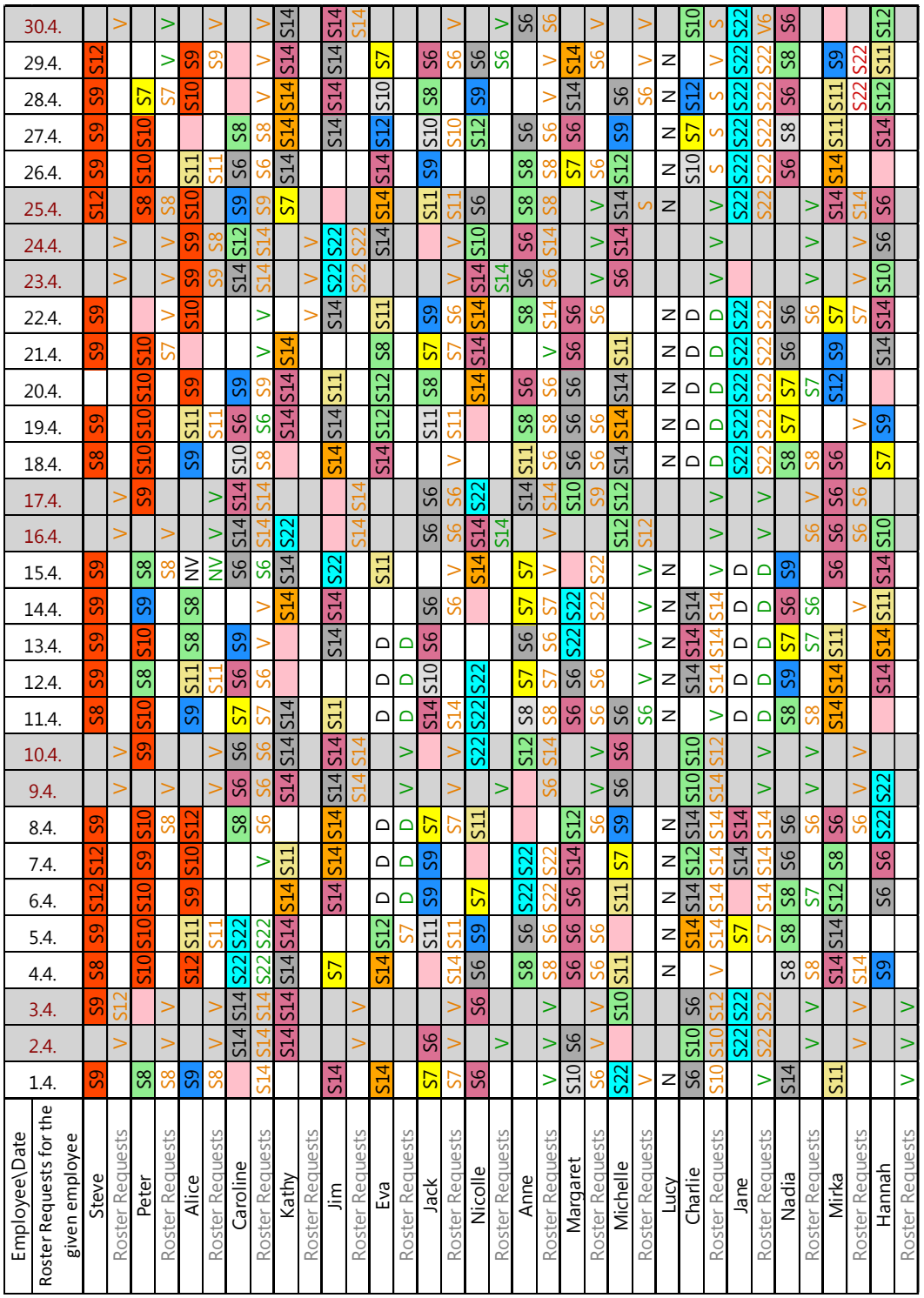


Figure 29: Generated Roster

We will now go through all attributes that influence the resulting rosters. First of such things are the parameters determining the importance of various roster features. We have defined these parameters in Chapter 3. Here we present values used for computation at customer's site that reflect the importance of various features to the customer.

UnderstaffingFactor	50	WantToWorkRequestCoefficient (WTW_coeff)	1
WantToWorkViolationFactor (WTWviolationFactor)	50	EmployeeActivityPriorityFactor (EAP_multiplier)	0.1
OptionalResourceRequirementFactor (ORRfactor)	0.1	EmployeeTimePreferenceFactor (ETP_multiplier)	0.1
DaysOffGroupingCoeficient	0.1	AllowUnsatisfiedWorkRequests	false
DaysOffGroupingCoeficient2	0.1	GroupingDaysOff	false
TimeOffRequestCoefficient (TOR_coeff)	1	PenalizingSingleDayOff	false
TORCWT_multiplier	0.5	ProhibitEmergencyFolloweByS22	true

Figure 30: Overview of Model Parameters

Now we have to discuss the fulfillment of hard constraints (non-optimizing features of the roaster). All the following statements can be manually verified against the Generated Roster. We will only present verification of features that can be expressed with a compact table or scheme. We will not prove the satisfaction of demand constraint, inter-shift breaks, weekly rest, contract working time enforcement, shift and activity limits, 'no emergency after six shifts' or 'no night shift after emergency'. All these features have been checked automatically by a revision tool. As an example of how we would proceed if we were to verify these conditions by hand we check the fulfillment of employee-shift limits.

Roster				Limits		
Employee	Shift	Actual #	≤	Max	Shift	Employee
Anne	14	1	✓	4	14	Anne
Anne	22	2	✓	2	22	Anne
Caroline	6	6	✓	6	6	Caroline
Caroline	14	5	✓	6	14	Caroline
Caroline	22	2	✓	2	22	Caroline
Charlie	6	2	✓	6	6	Charlie
Charlie	14	6	✓	6	14	Charlie
Eva	14	6	✓	6	14	Eva
Hannah	6	4	✓	6	6	Hannah
Hannah	14	6	✓	6	14	Hannah
Hannah	22	2	✓	2	22	Hannah
Jack	6	6	✓	6	6	Jack
Jack	14	1	✓	6	14	Jack

Roster				Limits		
Employee	Shift	Actual #	≤	Max	Shift	Employee
Kathy	22	1	✓	2	22	Kathy
Margaret	14	3	✓	4	14	Margaret
Margaret	22	2	✓	2	22	Margaret
Michelle	6	5	✓	6	6	Michelle
Michelle	14	5	✓	6	14	Michelle
Michelle	22	1	✓	2	22	Michelle
Mirka	6	5	✓	6	6	Mirka
Mirka	14	6	✓	6	14	Mirka
Nadia	14	1	✓	4	14	Nadia
Nicolle	6	5	✓	6	6	Nicolle
Nicolle	14	6	✓	6	14	Nicolle
Nicolle	22	4	✓	6	22	Nicolle
Jim	22	3	✓	6	22	Jim

Figure 31: Shift Limits Verification

Further we will focus on the degree of fulfillment of soft constraints. Let us start with **time off requests** as a vital feature. There is a total number of 61 requests with *auto* status, from which 49 were satisfied (80%), 7 were partially satisfied by scheduling emergency and 5 requests haven't been satisfied at all. The majority of unsatisfied/partially satisfied requests are conflicted with another request and as a necessary outcome due to lack of employee capacity some of them end up

unsatisfied. The following figure shows the TORs satisfaction breakdown. If we use the same formula as we have used in the model (TOR satisfied by emergency shift counts half of the fully satisfied TOR to the objective value) the average satisfaction of TORs per employee is 86.74%, and average number of 91.26% TORs are satisfied on average in one of ways.

TOR category\Employee	Steve	Peter	Alice	Caroline	Kathy	Jim	Jack	Anne	Margaret	Michelle	Charlie	Jane	Nadia	Mirka	Total
TORs handed in	8	7	5	5	3	2	9	4	3	3	2	1	1	8	61
TORs satisfied	8	5	5	2	3	2	6	3	1	2	2	1	1	8	49
TOR sat. by emerg.		2		2			2		1						7

Figure 32: TORs Satisfaction Breakdown

Another category of frequently entered requests are **want-to-work requests**. WTW requests with status set to *approved* are obviously satisfied in the optimal solution which can be easily verified visually from the roster.

There are a total number of 136 requests with *auto* status from which 107 were satisfied (78.67%) and 29 were not. It happens quite often that there are more requests for a certain shift on a single day than the total number of shifts supported. This way we end up with a certain number of requests impossible to be satisfied. The following figure shows the WTW requests satisfaction breakdown. The average satisfaction ratio for a single employee is 68.46%. When talking about approved WTW requests, there was no need to use a dummy variable in order to revert the infeasibility caused by some WTW request that has no (O)RR paired.

	Steve	Peter	Alice	Caroline	Jim	Eva	Jack	Anne	Margaret	Michelle	Charlie	Jane	Nadia	Mirka	Total
TTL WTWs	1	6	8	17	7	1	14	21	14	3	16	16	6	6	136
WTWs Satisfied	0	4	6	14	7	0	12	16	9	3	13	15	4	4	107

Figure 33: WTW Requests Satisfaction Breakdown

In case of this customer, the **fair policy (FP)** has eventually not been used for generating rosters. The problem is that FP approach is a direct opposite of employee-preference-based rosters. Due to the large scale tradeoff between assigning shifts uniformly (FP) and respecting irregular shift distribution (preferences) we gain on computational intensity and experience convergence issues with both FP and preferences with high objective function multipliers in place. However, in case of a

customer's site with lower level of employee request support the FP approach would gain perfect results.

The next feature we might want to analyze are the **employee-activity priorities**. There is actually just the supervisor activity involved. In case of **employee-time preference**, several employees have stated some preference on couple of shifts. The following figure compares the preference and the actual shift assignment in the real roster.

Employee	Eva	Jack			Nicolle	Anne	Alice	Kathy			Jim			Peter			Michelle	Jane				Nadia		Mirka	Steve	
Shift Start	6	6	12	22	22	6	8	6	8	12	6	14	22	6	8	11	22	6	8	14	22	6	12	10	8	12
Preference	-1	1	-1	-1	0.5	1	1	-1	-1	1	-1	1	0.5	-1	1	-1	-1	-1	-1	0.5	1	1	-1	-1	1	1
Percentage	0%	25%	0%	0%	17%	30%	9%	0%	0%	0%	0%	63%	13%	0%	17%	0%	4%	0%	0%	11%	68%	38%	0%	0%	14%	19%

Figure 34: Time Preference Fulfillment

We can see that the negative preference works very well as there was just one such shift assigned to an employee. With the positive values of time preference we can see quite good results. The only employee who seems not to respond well to these preferences is Alice. She is a supervisor with preference on supervisor activity. As such she has to cover the compulsory demand for supervisors (S9s) and therefore cannot satisfy the preference to the full extend.

The last quality of the roster we will look at are the **optional resource requirements (ORRs)**. The priority attribute is meant to model the order in which the ORRs are going to be used. We cannot express the wish to assign shifts in some sequence explicitly (without changing the solver) – we only can motivate the solver to do so with objective function's coefficients. The following figure shows the distribution of roster items that are based on ORRs. We can see satisfactory results.

Activity	Time			Priority	Quantity
Supervisor	10	-	18	1	31.4%
Operation	12	-	20	1	27.5%
VIP	12	-	20	0.8	5.9%
Supervisor	12	-	20	0.8	11.8%
Not Specified	10	-	18	0.6	11.8%
Not Specified	8	-	16	0.5	5.9%
Not Specified	9	-	17	0.4	0.0%
Not Specified	11	-	19	0.3	3.9%
Supervisor	9	-	17	0.3	2.0%
Not Specified	12	-	20	0.2	0.0%
Supervisor	10	-	18	0.1	0.0%

Figure 35: ORR Roster Items Distribution

We have exhaustively analyzed one concrete instance of the solved problem class. The solution has been accepted by the customer. The generated roster implements all hard constraints introduced in Formal Description of the Solved Problem chapter and it fulfills the soft constraints very well. By the time of writing this paper, the application has been launched for three months on the customer's site and has been used for scheduling. During that time we have conducted extensive testing on permutations of typical input with consistent results that are very similar to the results presented in this chapter. We have also developed test data for testing extreme cases of usage of individual roster features. Additionally, several other customers have shown interest in the application which has already resulted in one contract.

Chapter 6. Conclusion

In this paper we have presented a workforce optimization problem based on a real life situation. We have analyzed it thoroughly and created detailed specification of functionality that had to be achieved. Then we have modeled the problem as mixed integer program, modeled all features using just linear (in)equalities and expressions. The model has been then implemented in C# using .NET framework and third party solver and integrated with the main web application. We have shown the instance of the problem that the software is expected to solve and presented results that were achieved by the system. We have developed system that meets all the requirements we have stated. Also, the software has already been deployed at customer's site and has been successfully used.

The software has proven its usefulness as it is possible to coordinate the data collection and automate the generation of roster. The automated roster generating process does not suffer from human factor errors and allows for effective resource allocation taking into account all various well defined preferences. The whole system of preferences and requests has been formalized, whereas before it was only vaguely defined and incorporated into the rosters by managers based on experience.

Modeling various features as a MIP we have faced limitations of the expressive power of linear (in)equalities which is a downside of the chosen app0072oach. On the other hand after having designed the model we were able to use powerful MIP solver to find the optimal solution to our problem.

We have developed a full-fledged rostering tool. There are two possible ways of further development that can be taken. Firstly, if we were to solve very large problem instances beyond the capabilities of our software, we could use the column generation approach to redesign parts of the solution. It would involve coming up with a special tailored algorithm to generate candidate solutions. However, we could still reuse portions of the model describing roster qualities.

Secondly, the rostering tool can become part of a larger workforce management tool for contact centers and other businesses with random workload that would involve workload prediction, simulation, staffing and scheduling as preliminary stages of computation before the actual rostering. This ultimate piece of software is very likely going to be build based on results of this work.

Bibliography

Achterberg, Tobias. 2007. *Constraint Integer Programming.* Berlin : TU Berlin, 2007. D83.

Bloechliger, Ivo. 2004. Modeling Staff Scheduling Problems. *European Journal of Operational Research.* 2004, 158, page 533-542.

Brigitte Jaumard, Frédéric Semet, Tsevi Vovor. 1997. A generalized linear programming model for nurse scheduling. *European Journal of Operational Research.* 1997, Vol. 107.

Brunner, Jens O. 2010. *Flexible Shift Planning in the Service Industry.* München : Springer Heidelberg Dordrecht London New York, 2010. ISSN 0075-8442.

Chang-Chun Tsai, Sherman H.A. Li. 2008. A two-stage modeling with genetic algorithms for the nurse scheduling problem. 2008.

Jonathan F. Bard, Hadi W. Purnomo. 2005. Preference scheduling for nurses using column generation. *European Journal of Operational Research.* 510–534, 2005.

Koole, Ger. 2007. Call Center Mathematics. *www.CCmath.com.* [Online] January 26, 2007. [Cited: 01 25, 2010.] www.math.vu.nl/obp/callcenters.

Margarida Moz, Margarida Vaz Pato. 2005. A genetic algorithm approach to a nurse rerostering problem. *Computers & Operations Research.* 667-691, 2005, Vol. 34.

Miller, Steven J. 2007. *An Introduction to Linear Programming.* Providence : Brown University, 2007.

S. Kirkpatrick, C. D. Gelatt, Jr., M. P. Vecchi. 1983. Optimization by Simulated Annealing. *Science.* 1983, Vol. 220, 4598.

Seyda Topaloglu, Irem Ozkarahan. 2010. A constraintprogramming-based solution approach for medical resident. *Computers &OperationsResearch.* 2010, Vol. Computers&OperationsResearch.

SeydaTopaloglu, Hasan Selim. 2010. Nurse schedulingusingfuzzymodelingapproach. *Science Direct.* Fuzzy Sets and Systems, 2010, Vol. 161.

Slim Abdennadher, Hans Schlenker. 1999. Nurse Scheduling using Constraint Logic Programming. 1999.

Staffing Multiskill Call Centers via Linear Programming and Simulation. **Cezik Mehmet Tolga, L'Ecuyer Pierre. 2008.** Montréal : GERAD and Département d'Informatique et de Recherche Opérationnelle, 2008.