Design and Development of a Programming Language for Constrained Resource Allocation

Design und Entwicklung einer Programmiersprache für Ressourcenallokation
Master-Thesis von Robert Giegerich
Tag der Einreichung:

1. Gutachten: Prof. Dr. Mira Mezini
2. Gutachten: Prof. Dr. Oliver Hinz
Betreuer: Dr. Sebastian Erdweg

TECHNISCHE
UNIVERSITÄT
DARMSTADT

Fachbereich Informatik
Software Technology Group
Fachbereich Recht- und
Wirtschaftswissenschaften
Electronic Markets

Design and Development of a Programming Language for Constrained Resource Allocation
Design und Entwicklung einer Programmiersprache für Ressourcenallokation

Vorgelegte Master-Thesis von Robert Giegerich

1. Gutachten: Prof. Dr. Mira Mezini
2. Gutachten: Prof. Dr. Oliver Hinz
Betreuer: Dr. Sebastian Erdweg

Tag der Einreichung:

Erklärung zur Master-Thesis

Hiermit versichere ich, die vorliegende Master-Thesis ohne Hilfe Dritter nur mit den angegebenen Quellen und Hilfsmitteln angefertigt zu haben. Alle Stellen, die aus Quellen entnommen wurden, sind als solche kenntlich gemacht. Diese Arbeit hat in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegen.

Darmstadt, den April 30, 2016

(Robert Giegerich)

Abstract

This thesis depicts how the design and development of a DSL for the teacher assignment problem is conducted in order to achieve a comprehensible interface to a performant solving backend. A hybrid solving approach with a SAT solver for efficiency satisfaction and an iterative local search with an optimizer is implemented. The domain specific knowledge allows for simple but effective neighborhood functions to be implemented. The different solver characteristics allow for a flexible constraint language for the problem specification and yield a promising performance. We test the comprehensibility of our DSL on the target audience of teachers that usually have to be considered non-programmers. The results indicate that the DSL is comprehensible enough to be used by non-programmers. Together the DSL and the solving process offer an interesting programmer-compiler-relationship as a model for a versatile decision support system.

Contents

List of Tables

List of Figures

List of Listings

1 Introduction

Every year, schools have to plan the upcoming term's teaching schedule. Every class has to be taught by exactly one teacher for the specified time in each subject assigned by the german education authorities. Teachers differ in qualification concerning subjects and class levels. They should not work more than their regular teaching capacity. Finding a solution complying with all these constraints at hand falls into the complexity class of NP-hard problems. Insights and tools from Operations Research and Constraint Programming theoretically offer methods for problem domains such as this allocation problem. Already implemented search and optimization algorithms can be reused from commercial or non-commercial libraries. Compared to manually searching for a solution, they offer an efficient and automated way of solving a complex problem. But the usage of these libraries depends on a representation of the problem that is understandable by a computer. In many situations the person in charge of solving a problem is not able to formulate a mathematical model of the faced problem. With a mathematical representation as presented in [22] and [3], a planner might achieve a solution by entering the problem data of his individual problem instance. But as it is an individual problem instance, there are also individual problem characteristics which were not anticipated in the formulation of the model. As a result a technically valid solution might be achieved, but the individual demands of the planner are ignored. Therefore we propose a domain specific language that addresses these two difficulties. The DSL shall facilitate (1) the entering of the problem data and (2) expressing constraints in terms of preferences for specific solutions like "Teacher X must teach Class Y in subject Z" or "The class teacher of a class should teach as many courses in his class as possible".

Contributions

The contributions of this thesis can be summarized as in figure 1.1:

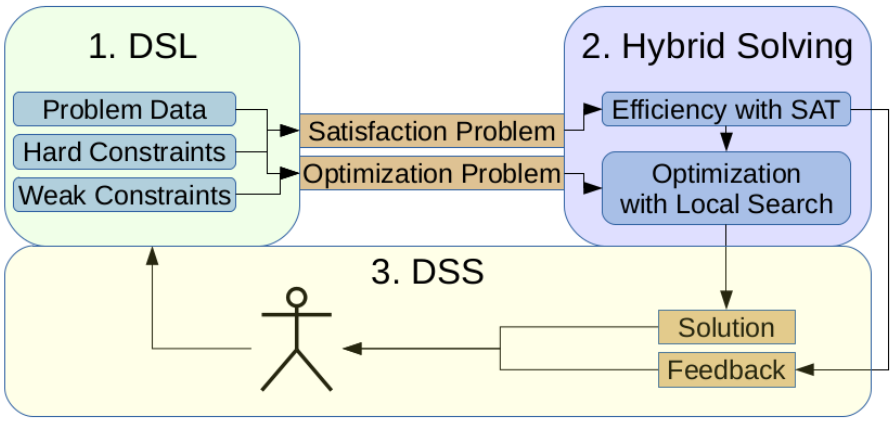

Figure 1.1.: Contributions

1. A DSL for the problem of teacher assignments and an interpreter that translates problem declarations into constraint satisfaction and optimization problems. Its constraint language concept allows very flexible problem formulations.

2. A hybrid solving process based on efficiency satisfaction and preference optimization by local search.

3. The proposal of a programmer-compiler-relationship as a flexible implementation model for a decision support system (DSS).

Structure

We introduce the specific problem domain at hand and the terminology of the scientific fields that address it in chapter 2. The design of our DSL is depicted in chapter 3 and its interpretation in chapter 4. The solving process is explained in chapter 5. An analysis of its performance can be found in chapter 6. We evaluate the comprehensibility of the DSL with a survey in chapter 7. We briefly discuss related and possible future work in chapter 8 and conclude in chapter 9.

2 Preliminaries

Speaking in economical terms lent from production theory, resource allocation deals with the combination of different input factors, which yield a specific output. The possible output is limited by the production technology that dictates which combinations are valid and what a specific combination yields. This simple model allows to state different decision problems.

1. Cost minimizing: Search for the least expensive combination of input factors yielding a specific output.
2. Maximization: Search for the most valuable output that can be created with a specific endowment of input factors.
3. Satisfaction: Search for any possibility to achieve a specific product with a specific endowment of input factors.

The scientific fields that address these combinatoric problems are Constraint Programming (CP) and Mathematical Programming (MP). CP's roots are in Logic Programming and Artificial Intelligence and therefore attributed to Computer Science. MP is commonly attributed to the fields of Operations Research (OR) and Applied Mathematics. Facing the same problems, they are overlapping in many areas. MP covers mainly optimization problems segregated into many problem classes. In CP the most common definition of a problem is the one of a Constraint Satisfaction Problem (CSP). It can be supplemented by an evaluation function and then very much resembles the general form of an optimization problem formulation in MP. For further distinction we refer the reader to [13]. We continue with considering MP to be more on a micro level, developing specific solutions/algorithms for specific problem classes, whereas CP covers a more holistic approach or as Rossi, Beek, and Walsh put it:

> "Constraint programming is 'programming' partly in the sense of programming in mathematical programming. The user declaratively states the constraints on the feasible solutions for a set of decision variables. However, constraint programming is also 'programming' in the sense of computer programming. The user additionally needs to program a search strategy." [20, p.3]

In the following sections we introduce the concrete problem domain (2.1) and deconstruct it into subproblems in order to be addressable with different search strategies on the macro level (2.2). Section 2.3 depicts the different tools and techniques we use in the implementation.

2.1 Problem Domain Teacher-Course-Assignment

The executives at a German public school know how many classes have to be taught and the state educational authority (Kultusministerium) dictates how many hours of which subject have to be taught in classes of a specific grade. The executive responsible for the task must search for an allocation of teachers to lessons that conforms with the educational requirements based on the employed teachers. The "production" of lessons has to be satisfied and the constraintness of the resource teacher results from labour regulations. The common target workload or capacity of a teacher is about 26 hours. An acceptable deviation of that amount is usually one to three hours, depending on former accumulated overtime or work reduction. When the planner cannot find a solution that lies within an acceptable area of deviation, the planner has an interest to assign teachers in a way that the deviation is as little as possible. The constraint satisfaction problem becomes a problem of optimization, meaning a search for a solution that is breaking the stated constraints, but only as few as possible. In addition to juridical

or contractual based constraints, the planner wants to assign specific teachers to specific classes. She also tries to follow rules such as assigning the class teacher to as many lessons possible in the specific class. These solution preferences can be used to evaluate the quality of the staff's assignments. When no acceptable solution can be found, the school faces a much more complex problem: In that case, the school may request funding for additional staff members with the necessary subject training. This introduces the possibility to optimize staffing regarding budget aspects. However, this possibility is out of scope for this thesis.

2.2 Problem Formulations

Depending on the current priorities and situation the planner finds herself in, either the CP approach with a CSP or the MP approach with an optimization problem might be more suitable. In the next sections we introduce the constituting components of both problem types and how they differ.

Constraint Satisfaction problem

A constraint satisfaction problem consists of a set of decision variables and a set of constraints. Decision variables can take values defined in their domain consisting of two or more values. Commonly the decision variables are Boolean variables with the domain `[true, false]`. Constraints state that some subsets of decision variables cannot be selected at the same time. They also can be seen as functions that map a set of decision variables to a Boolean value, meaning the constraint is satisfied or not. A solution to this problem is an assignment of `true` or `false` to all the decision variables while all constraints are satisfied. The problem is called satisfiable when such a solution can be found and otherwise unsatisfiable (see [2]).

Optimization problem

In optimization problems the domains of decision variables differ for different problem classes, the most common ones are continuous and discrete domain intervals. In addition to decision variables an optimization problem generally consists of an objective or target function $f : \Omega \rightarrow R$ that shall be minimized or maximized. Minimization (maximization) means the search for a solution $x \in \Omega$ so that for every $y \in \Omega$ the relation $f(x) \leq f(y)$ (respectively $f(x) \geq f(y)$) holds. The target function is subject to a constraint set as in the satisfaction problem. Again, the set of constraints dictates which decision variables can have which values in a solution. In addition to the validity of a solution candidate, the target function evaluates its quality, by including specific subsets of decision variables in the function (see [13]). In a minimizing problem a preferable decision variable x_i would be included with $f(x) = \ldots - x_i$, meaning that the selection makes the target function value smaller. These contributions to the target function refer to *soft constraints* in contrast to *hard constraints*. Soft constraints generally make the same statements about a specific subset of decision variables as hard constraints with the difference, that they are not mandatory during the search for a solution. Only their representation in the target function influences the solving process. Besides the inclusion of a penalty term into the objective function soft constraints can be implemented e.g. by stating that at least a specific amount of a subset of hard constraints must hold [16].

2.3 Solving Technology

For either optimization or satisfaction problems there are a number of solvers to choose from. Depending on what kind of a solution we want, a different formulation and therefore a different solving technology

has to be used. In the following section we will compare the problem formulations that are given to SAT solvers, Pseudo-Boolean solvers and solvers targeted at the different problem classes attributed to MP. In section 2.3 we compare the CP tool MiniZinc [19] to the common tools for MP that are used for optimization problems. Section 2.3 introduces SAT solvers and propositional formula that address a very strict form of a CSP. A compromise regarding expressiveness and efficiency seem to be Pseudo-Boolean solvers that are introduced in section 2.3.

Mathematical Programming

For optimization problems there are many problem classes e.g. Linear Programming (LP), Integer Linear Programming (ILP), Mixed Integer Linear Programming (MILP) and Mixed Integer Nonlinear Programming (MINLP) in ascending order of complexity. They have different demands on the form of the constraints and the target function and therefore every class has often similar variants of solving approaches but also with unique specialities. Due to these different problem classes in mathematical programming a variety of solvers exist. Algebraic Modeling Languages (AML) are domain specific languages that act as a common interface for different solvers. D'Ambrosio and Lodi mention AMPL [1] and GAMS [11] as the most widely used AML. In AMPL an optimization problem might look like shown in listing 2.1:

```
1  # Variables:
2  var x1 integer;
3  var x2 integer;
4
5  # Target function:
6  maximize z: 400*x1 + 50*x2;
7
8  # Constraints:
9  subject to Domain_x1: 0 <= x1 <= 70;
10 subject to Domain_x2: 0 <= x2 <= 500;
11 subject to Resource1: 25*x1 + 10*x2 <= 5000;
12 subject to Resource2: 100*x1 + 20*x2 <= 8000;
```

Listing 2.1: AMPL Example

In this example two integer decision variables are introduced and the target function `z` shall be maximized. The domains of `x1` and `x2` are limited by constraints in lines 9 and 10. Resource constraints in lines 11 and 12 introduce interdependency between the variables, e.g. that assigning a higher value to `x1` limits the maximum value one can assign to `x2`.

In contrast to AMPL or GAMS, MiniZinc is not an AML but described as a standard modelling language for CP [18]. Its features encompass algebraic modelling as well as the whole feature set of constraint programming, e.g. conditional constraints. This supports our understanding of CP being a broader field including MP. For the modelling of a problem in AMPL the programmer has to know about the solver that should be fed with the problem, due to solver specifics that influence how a problem can or should be formulated. For MiniZinc, solver independence is an explicit goal that is achieved with two problem formulation stages. In the first stage the modelling language MiniZinc is used by the programmer to formulate the model. In the second stage the MiniZinc formulation is translated into the low-level language FlatZinc [1]. FlatZinc is the interface language that a solver has to adopt in order to be accessible via MiniZinc. The translation of different model formulations for the same problem to a canonical form in FlatZinc leaves great freedom to the programmer. The solver will be fed with what was meant (canonical FlatZinc) rather than how it was formulated (flexible MiniZinc). For example one can use either Boolean decision variables or integer decision variables with the domain $[0, 1]$ to model the assignments of teachers to lessons. In case of Boolean decision variables the MiniZinc pipeline will translate them in a way that a solver only has to deal with the integer synonyms of the Boolean decision variables.

[1] http://www.minizinc.org/downloads/doc-1.6/mzn2fzn.pdf

SAT Solvers

Input for SAT solvers is propositional logic in conjunctive normal form (CNF). CNF propositions are of the following form [2]:

- $\bigwedge$: a conjunction of clauses
- $\bigvee$: a clause as a disjunction of literals
- $v, \neg v$: a literal being either a Boolean variable v or its negation $\neg v$

Literals correspond to decision variables with the domain `[true,false]`. Clauses are constraints on sets of literals. The conjunction expresses the set of constraints, i.e. every clause (single constraint) must hold. A SAT solver finds such a problem satisfiable when every literal can be assigned to either `true` or `false` so that all clauses and conjunctions evaluate to true. When such a combination of assignments cannot be found, the problem is found unsatisfiable. For most SAT solvers the CNF DIMACS text format is used. A small CSP in CNF DIMACS is shown in listing 2.2:

```
1 p cnf 5 3
2 1 -5 4 0
3 -1 5 3 4 0
4 -3 -4 0
```

Listing 2.2: CNF DIMACS Example

The first line introduces the problem as in CNF format with five decision variables and three clauses. Every following line is a single clause in which a positive integer refers to a `true` literal, a negative integer to a `false` literal, and `0` signals the end of the clause. The first clause in line 2 means $x_1 \vee \neg x_5 \vee x_4$. In contrast to AML the general expressiveness is considered to be higher, but problems arise for certain kinds of constraints such as a XOR constraint like $x_1 \otimes \ldots \otimes x_t$. In an AML we could simply state the arithmetic expression $\sum_{i=1}^{t} x_i = 1$. For a representation in propositional logic a translation has to be done that results in the following propositions:

$$x_1 \vee \neg x_2 \vee \ldots \vee \neg x_t \tag{2.1}$$
$$\neg x_1 \vee x_2 \vee \ldots \vee \neg x_t \tag{2.2}$$
$$\vdots$$
$$\neg x_1 \vee \neg x_2 \vee \ldots \vee x_t \tag{2.3}$$

Every single clause is needed to state that only one of the decision variables can be assigned to `true` at a time. Different SAT solvers implement such translations for different concepts implicitly. A cardinality constraint may be used for equation 2.1. It is of the form $x_0 + x_1 + \ldots + x_n = C$ where x_i is a literal that evaluates to 1 if it is a true literal or 0 if it is a false literal and $C \in [1, \infty]$.

Pseudo-Boolean Solvers

Pseudo-Boolean solvers (PB solvers) are generally considered a distinct technology for the distinct class of Pseudo-Boolean optimization problems. Opposed to the satisfiability problem a target function is stated. A Pseudo-Boolean constraint is similar to cardinality constraints. They describe constraints where decision variables should contribute to a different extent to the left-hand side of a cardinality constraint. With w_i being the weight of a decision variable, a Pseudo-Boolean constraint is written analoguously to the cardinality constraint as $\sum_{i=1}^{t} x_i * w_i = C$. A Pseudo-Boolean constraint is satisfied, when the sum

of its left-hand side is greater or equal to the right-hand side. Though the general form is very close to the problem formulations in AML, a great part of research and development of PB solvers is targeted at the links to SAT solvers. In [9] Eén and Sorensson suggest a translation of PB constraints into SAT constraints and a native use of a SAT solver. This is the opposite approach to the alteration of SAT solvers to natively support PB constraints that is used in several solvers like Sat4j [15], which we use in our solving process.

2.4 Solving Strategies

Although the different solvers differ in specific algorithmic implementation there are activities that are common to all of them. The two main activities during the solving of the forementioned problems are *search* and *inference*. Every solver implements them on a micro level and we implement them on a macro level with our solving process design.

Search

Search refers to the traversion of the search space that contains all the solution candidates and eliminating subspaces with trial and error. When a solution has been explored and found invalid, other solution candidates with the same characteristics leading to invalidity will not be explored (see [10]).

Local Search

Applying local search means to take an arbitrary solution candidate that may even be infeasible or incomplete and try to enhance it by iteratively modifying it. Local in this context means that only a small neighborhood of solution candidates will be explored. This neighborhood is calculated by application of a neighborhood function (see [14]). A simple neighborhood function might be given a subset of decision variables and return all solution candidates that result from the permutations of the possible values for the subset. Depending on the number of generated solution candidates, a complete search on this neighborhood is feasible and a local optimum can be found.

Inference

Inference refers to the reasoning about the constraints, resulting in a smaller search space. In CP a key feature in terms of inference is constraint propagation through consistency. Consistency between two or more constraints is achieved by examination of the effects of one constraint onto the domains of the decision variable relevant to other constraints (see [10]). As an example take the following two clauses:

1 x_1
2 $\neg x_1 \vee x_2$

The clause in line 1 requires the decision variable x_1 to be assigned `true`. The clause in line 2 constitutes an implication meaning if x_1 is assigned `true`, also x_2 has to be assigned `true`. Both clauses combined let us infer that because the domain of x_1 is reduced to `[true]`, the domain of x_2 is also reduced to `[true]`. Propagation then means to “inform” other constraints of this domain limitation which again can induce further propagation or even result in a conflict when a domain becomes empty. With this reasoning large subspaces of the complete search space can be identified as not containing a valid solution.

3 Specification Language

Our programming language is a DSL for school executives hence usually non-programmers. Therefore our design should reflect a healthy balance between comprehensibility for non-programmers and expressiveness in terms of functionality. The implementation of the DSL is done with the language workbench Spoofax in the Syntax Definition Formalism SDF3 (see [17]). In this chapter we depict the implementation of the language. In section 3.1 we show the different DSL expressions by example and thereby introduce the underlying object model of our DSL and the constraint language. In section 3.2 we show the design of non-trivial concepts of our DSL.

3.1 Problem Specification Syntax

A problem specification is segregated into different sections. For every section we show what can be expressed and what its semantics are.

Subjects

The subjects section contains all subjects of the specific school. They have to be explicitly listed in the problem specification. This allows us to implement a reference check anywhere where subjects have to be referenced in order to avoid misspellings. Teachers usually have to have gained a specific qualification for each subject in order to teach it. There are some lessons that teachers can also give without a specific qualification e.g. a special coordination lesson that are of an administrative nature. Other lessons don't need one specific subject qualification but one of a set of subject qualifications.

```
1 Subjects
2   Subject english
3   Subject mathematics
4   Subject german
5   Lesson dyslexiasupport with possible subject qualifications: deutsch,english
6   Lesson without subject qualification coordination
```

Listing 3.1: Subject Declaration

In 3.1 the unique subjects `english`, `mathematics` and `german` are declared to require a later declared lesson to be taught by a teacher with the corresponding subject qualification. A special lesson is `dyslexiasupport`, which does not require a specific subject qualification but can be taught by teachers with `german` or `english` qualification. The subject `coordination` can be referenced for lessons where the individual qualification of a teacher doesn't matter.

Teachers

The teachers section contains every employed teacher. A teacher is declared with a unique ID and the relevant information regarding the staffing decision: the target workload and the subject qualifications.

```
1 Teachers
2    Teacher One
3     Career : GYM
4     Subject qualifications :
5       Gymnasium: german,ethics,economics
6     Target workload : 26
7    Teacher Two
8     Career : HR
9     Subject qualifications :
```

```
10      Haupt-/Realschule: english,pe
11    Target workload : 25,5
```

Listing 3.2: Teacher Declaration

In listing 3.2 teacher `One` can teach the subjects `german`, `ethics` and `politics` all on Gymnasium level and should have a workload of 26 hours. Teacher `Two` teaches `english` and `pe` on Haupt-/Realschule level and should work 25.5 hours.

Lessonplan

In the lessonplan section the lessons that are common to classes of an equal grade are declared. The lessonplans are implicitly referenced by class declarations.

```
1  Lessonplans
2   Lessonplan Realschule grade 8 :
3     german -> 4
4     english -> 3
5     mathematics -> 4
6     classteacher -> 0
```

Listing 3.3: Lessonplan Declaration

In listing 3.3 the lessonplan for every class in grade 8 in the school path Realschule is declared. All of the concerned classes each have 3 hours of english lessons, 4 hours of `german` and `mathematics` lessons and are assigned a teacher as `classteacher` that does not result in actual workload (0 hours).

Workload

There are four categories of workload. There is a category for each different level (Haupt-, Realschule, Gymnasium and Oberstufe) and a category for special assignments that are not lessons like the principal's workload or library service. Special assignments are usually preset by the planner, so each of them is declared with a unique ID, its workload and the ID of the assigned teacher. In the other three categories classes and courses are declared.

```
1  Special assignments
2       Principal : 22 : teacherX
3       Library : 3 : teacherY
4
5  Realschule
6      Class 8a
7      Class 8b with deviation from lessonplan:
8        deutsch -> 5
9        mathematik -> 5
10       englisch -> 3 with 2 teachers
11     Class 8c
12     Common course for classes 8a,8b,8c:
13       ethics -> 2
14       religion -> 2
15       french -> 1
16 Oberstufe
17     Grade 12 :
18           Basic course: german -> 2
19           Basic course: german -> 2
20           Basic course: german -> 2
21           Advanced course: german -> 5
```

Listing 3.4: Workload Declaration

In 3.4 teacherX is assigned to special assignment `Principal` and teacherY is assigned to special assignment `Library`. In category `Realschule` classes 8a, 8b and 8c implicitly refer to the declaration of workload from their respective lessonplan (Realschule Grade 8) depicted in listing 3.3. Instead of four

hours of german and mathematics class 8b requires five hours. Class 8b also requires an additional teacher for the three hours of english. The three classes 8a, 8b and 8c all share courses in ethics and religion with two hours and one course with one hour of french. In the section Oberstufe the single courses are listed individually as either basic or advanced course, as shown for three basic courses with 2 hours of german and one intensive course in german with 5 hours.

Constraints

In the constraints section the planner can declare preferences that a solution to the declared problem must or should fulfill. We distinguish between hard constraints and weak constraints. Hard constraints must be fulfilled by a solution in order for the solution to be considered valid. Weak constraints only influence the solving backend to a certain extent, meaning that we encourage the solving backend to adhere to a weak constraint. As the solving backend includes all weak constraints at once into its decision making, we provide each weak constraint with a weight in order to weigh constraints against each other.

```
1  Constraints
2     Weak constraint with weight 4:
3       Forbid: teacher A teaches classes in grade 9
4
5     Weak constraint with weight 5:
6       Count of assignments where teacher A teaches subject french >= 2
7
8     Hard constraint:
9       Assign: teacher C teaches class Gymnasium 7b in subject history
10
11    Hard constraint:
12      Overtime of teacher D = 0
13
14    Hard constraint:
15      For each teacher X in teachers with qualification pe:
16        Workreduction of teacher X <= 3
17
18    Hard constraint:
19      For each teacher X in teachers with qualification history-bilingual
20        and each class Y in classes with subject history-bilingual:
21          If teacher X teaches class Y in subject history-bilingual,
22           then teacher X teaches class Y in subject english
```

Listing 3.5: Constraint Declaration

In 3.5 the weak constraint at line 2 has weight 4 and says that teacher A should not teach classes in grade 9. The weak constraint at line 5 has weight 5 and says that teacher A should teach french in at least two courses. Given the case that the subject french is only taught in 9th grade, the solving backend should assign teacher A to those classes, as the french-teaching-constraint is declared with a higher weight. The hard constraint in line 8 presets the assignment of teacher C to class 7b at the Gymnasium level in subject history, given this assignment is a valid one, meaning class 7b needs history lessons and teacher C has a qualification for history at level Gymnasium. The hard constraint at line 11 requires to find a solution where teacher D is assigned lessons amounting to exactly his target workload. The hard constraint at line 14 requires to find a solution where teacher X is assigned a workload at least equal to her target workload reduced by 3. At this point teacher X is an iterator over the set of teachers with a pe qualification. Hence the reduction constraint applies to all pe teachers. The hard constraint at line 18 is introducing a teacher iterator over the set of all teachers with a *history-bilingual* qualification and a class iterator over the set of classes that need history-bilingual lessons. The If-then constraint in line 21 forces the solver to assign teacher X to class Y in subject english, if teacher X is assigned to class Y in subject history-bilingual.

Deputations

Deputations describe effects of solutions on target workloads of teachers. For example some lessons are considered to require more engagement than others. When a certain amount of such lessons is assigned to a teacher, the actual workload caused by these lessons should count more, e.g. reduce the target workload of the teacher.

```
1 Deputations
2     Deputation Oberstufe for teacher X when Workload of assignments where teacher X teaches classes
          in grade 12,13 >= 7
```

Listing 3.6: Deputation Declaration

The deputation at line 2 in listing 3.6 is named `Oberstufe`. It is assigned to each teacher whose assignments contain seven or more hours of lessons in courses in grade `12` and `13`.

3.2 Spoofax Design

In this section we describe the Spoofax implementation of our DSL. It is best understood in a top-down explanation. Therefore we begin with the architecture and reason about implementational details where they are non-trivial or where there are implications between architecture and implementational details. As Spoofax allows separation of different parts of a language into different modules, we made use of it and separated our DSL into four modules. The module Common ships with a fresh Spoofax example project and defines low-level productions like character classes. The module Data defines the basic object model that encompasses subjects, teachers and classes. As its implementation is very straightforward and exemplified in section 3.1, we will not discuss it in detail. The module Constraints holds the concepts for declaring constraints and rules for the solving backend to follow by. The central module TeachAlloc imports the modules Common, Data and Constraints.

TeachAlloc

The module TeachAlloc defines the structure of a problem as follows in listing 3.7:

```
1  Problem ::= "problem" ⟨ID⟩
2      ⟨Subjects⟩
3      ⟨Teachers⟩
4      ⟨Lessonplans⟩
5      ⟨SpecialAssignments⟩?
6      ⟨Haupt⟩?
7      ⟨Real⟩?
8      ⟨Gymnasium⟩?
9      ⟨Oberstufe⟩?
10     ⟨Constraints⟩?
11     ⟨Deputations⟩?
```

Listing 3.7: Problem Production

There are the three mandatory sections for subjects, teachers and lessonplans and six optional sections for special assignments, the four different school paths and the constraints. The contents of each section are already exemplified in chapter 3.1, therefore we will not discuss them in detail. Except for the Constraints and Deputations sections in all optional sections the actual workload to be distributed among teachers is declared. Every school path section defines its own scope in terms of class identifiers since the naming conventions are the same for every school path except for the section Oberstufe. Naming conventions are enforced with a special sort that limits class identifiers to be constructed according to the character classes in listing 3.8.

```
CLASSNAME ::= ⟨GRADE⟩ ⟨SHORT⟩
GRADE ::= [1-9]
       | [1][0]
SHORT ::= [a-z]
```

Listing 3.8: Class Naming Convention Production

Section SpecialAssignments and Oberstufe contain different productions from the ones in Haupt, Real and Gymnasium for the declaration of workload as depicted in section 3.1. In the Oberstufe path there are no classes anymore, students are assigned only to courses according to individual schedules. Here we only differentiate basic and advanced courses and they correspond to a single subject.

Constraints

In the Constraints module we implement the varieties of constraints we have shown in section 3.1. Their design is guided by the goal to be as closely to natural language as possible. At the same time we want to achieve an efficient design, following design principles like low redundancy and re-use. We identified two central concepts necessary for such an efficient implementation of a constraint language. First we need to express the different varieties of constraints as closely to the users domain language that is his natural language. Secondly we need to reflect their semantics in terms of which relationship between decision variables they should express. This leads us to the implementation of a query language component that is aimed at retrieving the different sets of decision variables that need to be referenced by the different constraint varieties. An abstraction for these sets is the ConstraintDomain.

ConstraintDomain

```
1 ConstraintDomain ::=
2  ⟨TeacherSelector⟩ "teaches" ⟨ClassSelector⟩ "in" ⟨SubjectSelector⟩
3  | ⟨TeacherSelector⟩ "teaches" ⟨SubjectSelector⟩
4  | ⟨TeacherSelector⟩ "teaches" ⟨ClassSelector⟩
```

Listing 3.9: ConstraintDomain Production

Listing 3.9 shows different productions for a ConstraintDomain. A ConstraintDomain refers directly to decision variables. It targets all the decision variables that match the corresponding TeacherSelector, ClassSelector and SubjectSelector with all three properties (teacher, class, subject). The most general constructor (line 2) takes one selector for each decision variable property and the other two are abstractions for not applying a filter on the class property (line 3) or the subject property (line 4). The selectors are implemented to express the desired segregation of teachers, classes and subjects. For teachers we want to be able to select individual teachers by name, all teachers, teachers with a certain subject qualification and the teachers on a specific career path. Listing 3.10 shows the constructors for a TeacherSelector in this order.

```
1 TeacherSelector ::= "teacher" ⟨ID⟩
2  | "all teachers"
3  | "teachers with qualification" ⟨ID⟩
4  | "teachers with career" ⟨ID⟩
```

Listing 3.10: TeacherSelector Productions

For classes we want a similar segregation. But since the naming conventions in schools are the same for every school path and the name of a class is only unique in its schoolpath, every selector referring to a specific class needs to include the information about the specific school path as shown in listing 3.11.

```
1 ClassSelector ::= "class Gymnasium" ⟨CLASSNAME⟩+
2   | "class Realschule" ⟨CLASSNAME⟩+
3   | "class Hauptschule" ⟨CLASSNAME⟩+
```

```
4   | "all classes"
5   | "classes with subject" ⟨ID⟩
6   | "classes in grade" ⟨INT⟩+
```

Listing 3.11: ClassSelector Productions

Constraints

Most of the several kinds of constraints were presented in section 3.1. In listing 3.12 we show how the concept of a ConstraintDomain fits to different constraint productions.

```
1 SetTrue ::= "Assign:" ⟨ConstraintDomain⟩
2 SetFalse ::= "Forbid:" ⟨ConstraintDomain⟩
```

Listing 3.12: SetTrue and SetFalse Productions

These varieties of a constraint express assignments and the prohibition of assignments. The ConstraintDomain with its different selector varieties allows to refer to single decision variables or sets of decision variables. The single components of a ConstraintDomain also contribute to the creation of rules as exemplified in listing 3.13:

```
1 TeacherForeach ::=
2 "For each" ⟨TeacherIterator⟩ "in" ⟨TeacherSelector⟩ ⟨TeacherLimiter⟩? ":"
3       ⟨Constraint⟩+
```

Listing 3.13: TeacherForeach Production

Here the TeacherSelector can refer to a set of teachers and the TeacherIterator will refer to the single teachers in the set and can be referenced in the constraints of this for-each-loop. The same concept applies to the iteration over a set of classes.

4 Problem Interpretation

Spoofax automatically creates an abstract syntax tree (AST) of a problem specification and passes it to our interpreter. The design of our DSL is purely declarative, meaning that there are no side effects. The compilation and interpretation of a problem declaration is therefore a mere translation from the AST to an object model implemented in Scala[1]. Our interpreter abstracts from the different varieties like classes, courses and special assignments, and encapsulates the information necessary to formulate the user's problem declaration as CSP and optimization problem. Only the declared constraints might be considered as having side effects. But these interactions only occur during solving, e.g. when two weak constraints conflict. The initial interpretation of a constraint is without side effects, too.

4.1 Domain Object odel

The object model is implemented in the abstract class TEACHALLOC, subclassed by the classes that implement the central concepts of the problem domain.

Problem

A succesful traversion of the AST results in an object of type Problem, holding all problem specification data. The entities are stored efficiently in Scala Maps if they need to be individually identified throughout the solving process. Hard constraints and deputations are stored in a Seq, and weak constraints in a Seq of pairs that hold the weak constraint and its weight as shown in listing 4.1.

```
case class Problem(val teachers: Map[String, Teacher], val lessonplans: Map[String, Lessonplan],
    val classes: Map[String, Schoolclass], val subjects: Map[String, Subject], val
    hardConstraints: Seq[Constraint], val weakConstraints: Seq[(Constraint, Int)], val
    deputations: Seq[Deputat]) extends TEACHALLOC
```

Listing 4.1: Problem Implementation

The traversion of the AST is implemented in the function `from`, depicted in listing 4.2. It first tries to match the AST as a "Problem" production and then matches every child node. The child nodes correspond to the different sections we allow to declare, like the Subject section and Teacher section. Every section should contain only a certain kind of production as child nodes. Therefore we call the `from` functions of the classes we want to translate the expected child nodes to.

```
def from(term: StrategoTerm, output: Any => Unit): Problem = {
 /* ... */
  term match {
    case StrategoAppl("Problem", StrategoString(name), children@_*) =>
      children.foreach { x => x match {
         case StrategoAppl("Subjects", subjects @ _*) => subjectList = subjects match {
           case Seq(StrategoList(subs)) => subs.map { x => Subject.from(x).id -> Subject.from(x)
               }.toMap
         }
         case StrategoAppl("Teachers", teachers @ _*) => teacherMap = teachers match {
           case Seq(StrategoList(ts)) => ts.map { x => Teacher.from(x).id -> Teacher.from(x) }.toMap
         }
         case StrategoAppl("None") => // Do nothing
         case StrategoAppl("Some", optionalSegments@_*) => optionalSegments.foreach { segment =>
             segment match {
    /* ...
     ...*/
```

[1] Scala is chosen due to its convenient features such as pattern matching, which facilitates the implementation of an interpreter.

```
16        case StrategoAppl("SectionGymnasium", classes @ _*) =>
17        var courseList = (classes match {
18        case Seq(StrategoList(cs)) => cs.map { x => (Schoolclass.from(x, GYMNASIUM, output).name ->
              Schoolclass.from(x, GYMNASIUM, output)) }
19        })
20      /* ...
21        ...*/
22      case StrategoAppl("Constraints", hards @ _*) => hards match {
23        case Seq(StrategoList(constraints)) => constraints.foreach { x =>
24          x match {
25            case StrategoAppl("Weak", StrategoInt(weight), const) => weakConstraintList =
                  weakConstraintList ++ Constraint.from(const, output).map { x => (x, weight) }
26            case StrategoAppl("Hard", const @ _*) => hardConstraintList = hardConstraintList ++
                  const.foldLeft(Seq[Constraint]())((acc, elem) => acc ++ Constraint.from(elem, output))
27          }}}
28      case StrategoAppl("Deputate", deputations@_*) => deputations match {
29        case Seq(StrategoList(deputate)) => deputationsList = deputate.map{ x => Deputat.from(x,
              output) }}}}
30          case StrategoAppl("Lessonplans", lessonplans @ _*) => lessonplanMap = lessonplans match {
31            case Seq(StrategoList(lps)) => lps.map { x => (Lessonplan.from(x).id ->
                    Lessonplan.from(x)) }.toMap
32          }}}
33          /* ... */
34        new Problem(teacherMap, lessonplanMap, classMap, subjectList, hardConstraintList,
              weakConstraintList, deputationsList)
35        }
```

Listing 4.2: AST Traversion

The pattern matching collects the problem entities in mutable collections in order to allow postprocessing. Postprocessing is necessary to ensure unique identifiability of classes. The collections are then used to instantiate the Problem instance.

WorkloadUnit

Objects of the class WorkloadUnit encapsulate the qualities of lessons and special assignments.

```
1  case class WorkloadUnit(subject: String, hours: Double, assignedTeacher: Option[String],
       numberOfTeachers: Int)
```

Listing 4.3: WorkloadUnit Implementation

In listing 4.3 the String `subject` denotes the subject qualification that is necessary to be assigned to this WorkloadUnit. The Double `hours` denotes the resulting workload for the assigned teacher. When a WorkloadUnit is already assigned per user declaration, the Option[String] `assignedTeacher` contains the assigned teacher. The Int `numberOfTeachers` denotes how many teachers shall be assigned to this WorkloadUnit.

Schoolclass

Objects of the class Schoolclass are an abstraction for the different groupings of students in the school. Every Schoolclass represents an unique group of students and contains the information about the workload units this specific group causes.

```
1  case class Schoolclass(val name: String, val schoolpath: String, val lessonplan: Option[String],
       val extraLessons: Option[Seq[WorkloadUnit]], val grade: Int, val category: String, val teacher:
       Option[String], val affectedClasses: Option[Seq[String]]) extends TEACHALLOC
```

Listing 4.4: Schoolclass Implementation

All workload declaration productions result in such an object. In listing 4.4 the String `schoolpath` denotes to which school path the Schoolclass is attributed. The Option[String] `lessonplan` contains either a String that refers to a Lessonplan or None to reflect that this Schoolclass has a fully individual

schedule. In addition to the Lessonplan the Option[Seq[WorkloadUnit]] `extraLessons` may contain additional workload on top of the lessonplan, or when the subject is listed in the lessonplan, it will be overwritten. The Int `grade` denotes the grade the Schoolclass is attributed to. In the Option[Seq[String]] `affectedClasses` the classes of all the students corresponding to this Schoolclass are listed. The function `getAllSubjectsRequired` returns the consolidated workload, e.g. the lessonplan combined with the extra lessons.

Teacher

Objects of the class Teacher represent the central resource of our decision problem.

```
1  case class Teacher(val id: String, val career: Int, val subjectQualifications: Map[String, Int],
       val targetWorkload: Double) extends TEACHALLOC
```

Listing 4.5: Teacher Implementation

As shown in listing 4.5, a teacher can be uniquely identified by the String `id` and has the general education for the career of a specific school path denoted by Int `career`. She has a target workload value equal to the Double `targetWorkload` and is allowed to teach the subjects contained in the Map `subjectQualifications` whose values denote the school level at which the subject can be taught.

Subject

Objects of the class Subject reflect the central quality of a workload unit.

```
1  case class Subject(val id: String, val needQualification: Boolean, val synonyms: Seq[String])
       extends TEACHALLOC
```

Listing 4.6: Subject Implementation

As shown in listing 4.6, a subject is uniquely identified by the String `id`, it corresponds to other subjects listed in the Seq `synonyms` and requires the teacher that is a assigned to a workload unit with this subject to have an explicit qualification denoted by the Boolean `requiresQualification`.

Lessonplan

Objects of the class Lessonplan reflect that classes in one grade usually share the same types of WorkloadUnits.

```
1  case class Lessonplan(val id: String, val plan: Map[String, WorkloadUnit]) extends TEACHALLOC
```

Listing 4.7: Lessonplan Implementation

The Map[String, WorkloadUnit] holds the subjects and their corresponding WorkloadUnit as key-value-pair. This implies that every subject can exist only once in a Lessonplan.

DecisionVariable

Central to the Problem object are the generated DecisionVariable objects that represent possible assignments of teachers to workload units.

```
1  case class DecisionVariable(val id: Int, val isSet: Boolean, val weight: Double, val teacher:
       String, val classes: Seq[String], val subject: String, val zincAddition: Option[String])
```

Listing 4.8: DecisionVariable Implementation

In listing 4.8 the Int `id` identifies the DecisionVariable by its (positive) DIMACS representation. It can either be `true` or `false` denoted by `isSet`. The `weight` denotes the hours the assignment would cause. The String `teacher` identifies the teacher the assignment refers to, the Seq[String] `classes` identifies all classes this assignment refers to and the String `subject` identifies the subject this assignment refers to. The naming conventions for the variables in the optimization formulation of our problem might result in multiple variables with the same identifier. The optional String `zincAddition` is appended to the identifier to preserve uniqueness.

Solution

Objects of the class Solution refer to a specific Problem instance `problem` and hold the assignments reflected by this solution in a Map[Int,DecisionVariable] `values` that corresponds to the one of the Problem instance as shown in listing 4.9.

```
1  case class Solution(val values: Map[Int, DecisionVariable], val problem: Problem)
```

Listing 4.9: Solution Implementation

There are several implemented functions that give information on the quality of the specific solution in terms of efficiency, e.g. the deviation from the target workload or the total overtime caused by this solution.

4.2 Problem Formulation

To exploit the advantages of the different problem formulations and respective solving technology, a Problem object can be approached either as a CSP or an optimization problem. We choose Sat4j as the solving library for the CSP formulation of a Problem instance. It is rather widely used, e.g. in the Eclipse IDE that is already a host for our DSL implementation with Spoofax. In addition, its feature set contains a good expressiveness due to implicit translations like implications and logical gates. The library offers Pseudo-Boolean capabilities and multiple solving strategies to experiment with. Although its performance compared to other SAT solvers is at best average [15], the PB solver performs at a good level. With MiniZinc we choose the framework that offers a very expressive modelling language for both CSP and optimization. The many solvers that can be used with it include GeCode, a very versatile and performant solver according to different competitions. For both technologies we use the same problem formulation that is based upon the decision variables we generate from the problem entities.

Sat4j Setup

With the instantiation of a Problem, we create an instance of a PB solver based on the SAT core of Sat4j and a DependencyHelper as shown in listing 4.10. Henceforth we will use PB solver and SAT solver synonymously. The DependencyHelper wraps around the solver and is used to programmatically formulate the problem. Interaction with a native solver of the Sat4j library is done by using the DIMACS format, e.g. referring to decision variables is done by positive and negative integer values. Therefore the mapping of a DecisionVariable to the DIMACS representation has to be handled explicitly, as we do by identifying DecisionVariable instances with a unique integer. The DependencyHelper would allow us to make the same statements with an arbitrary class of objects and internally map them to DIMACS.

```
1  val solver = SolverFactory.newDefault()
2  val depHelper = new DependencyHelper[Int, String](solver)
```

Listing 4.10: Sat4j Setup

The SolverFactory allows for different solvers to be instantiated, resulting in different solving behaviour and hence performance. The DependencyHelper translates constraints like implications into a simplistic form and imposes them onto the solver instance. Every constraint that is added via the DependencyHelper has to be given a name. That is because the DependencyHelper also offers to ask for reasons in case a solving run returned unsatisfiability.

MiniZinc Setup

MiniZinc is not accessible as a native Java or Scala library, but can be used by writing a problem declaration to a file. Therefore the setup to address our problem as optimization problem is the process of creating a String in the MiniZinc format, writing it to a file and invoking the MiniZinc binaries on it. We first call the `mzn2fzn` binary on the MiniZinc file which transforms it into the FlatZinc format and writes it to a separate file. This file can be handed to every FlatZinc interface of a solver. We use the binary `fzn-gecode` that ships with the MiniZinc installation. The different options available are discussed in chapter 5.

Problem Initialization

To prepare a Problem instance for both solving approaches, the `preprocess` function has to be called on it. It traverses the problem entities teachers, classes and lessonplans as shown as pseudo-code in listing 4.11 in order to generate the decision variables.

```
1 FOREACH WorkloadUnit X
2   FOREACH Teacher Y
3     IF Y.isQualifiedToTeach(X)
4       THEN createDecisionVariable
```

Listing 4.11: Decision Variable Generation

In the first step the individual demands are identified. By combination of classes, lessonplans and extra courses, we derive every workload unit. In the second step we search for all teachers that are qualified to be assigned to this workload unit. For every qualified teacher we create a decision variable that is uniquely identified by an integer ID. This ID is the DIMACS representation and is used to build constraints with the Pseudo-Boolean solver. There is one special case that has to be considered: In case the user has already assigned a teacher to a specific workload unit in the problem declaration we don't have to generate the decision variables for all teachers that are qualified to be assigned. By creating only the one decision variable that represents the assignment of the teacher declared by the user, we imply the assignment by reducing the search space for the solver. In the third step, we prepare the SAT solver by adding the problem domain's elementary constraints: how many teachers should be assigned to a specific workload unit. The number of teachers n_t is either user defined in the problem specification or by standard 1. Therefore we add two constraints with the DependencyHelper object to the solver:

```
1 depHelper.atLeast("Require at least " + n + " teacher(s) for assignment " + key, n,
      variables.toArray: _*)
2 depHelper.atMost("Require at most " + n + " teacher(s) for assignment " + key, n,
      variables.toArray: _*)
```

Listing 4.12: Imposition of Inherent Constraints

Listing 4.12 shows how the solver is instructed to find a solution that selects exactly `n` decision variables of the sequence `variables`.
In a next step shown in listing 4.13 the user defined constraints that are declared as hard constraints are added to the solver by the `apply` function of the Constraint trait that is introduced in section 4.3.

```
1 hardConstraints.foreach { constraint =>
```

```
2       constraint.apply(this, Map[String, String]())
3     }
```

Listing 4.13: Imposition of User Declared Constraints

Now the PB solver is provided with the complete search space and all the necessary hard constraints that have to be fulfilled. In case some of the hard constraints are contradicting each other, e.g. the solver finds a conflict, an exception is thrown and the user is advised to resolve the conflict.

4.3 Constraint architecture

The different types of constraints we want to use throughout the solving process implement the trait Constraint. It reflects the different forms a constraint can take in the different problem formulations (hard, weak, CSP, Optimization) and provides information on whether this constraint is fulfilled in a specific solution and gives information on which decision variables may be relevant for the fulfilling of this constraint:

```
1  trait Constraint {
2   def apply(problem: Problem, substitutionEnv: Map[String,String], output: Any => Unit)
3   def getOptimizationConstraint(problem: Problem, substitutionEnv: Map[String,String]) : String
4   def getTargetFunctionTerm(problem: Problem, substitutionEnv: Map[String,String], weight: Int) :
        String
5   def getFreeVars(solution: Solution, substitutionEnv: Map[String,String]) : Seq[Seq[Int]]
6   def isFulfilled(solution: Solution, substitutionEnv: Map[String,String]) : Boolean
7  }
```

Listing 4.14: Constraint Trait

- **`apply`** imposes the constraint as hard onto the PB solver instance of the Problem instance `problem`. This is done by selecting the relevant decision variables of `problem` and formulating them as a constraint. The environment `substitutionEnv` allows static scoping of class and teacher identifiers that will be explained in the next section.
- **`getOptimizationConstraint`** returns a String representation of the hard constraint in the MiniZinc language. Again the relevant decision variables of `problem` are retrieved and arranged as a String that enforces the MiniZinc pipeline to adhere to this constraint.
- **`getTargetFunctionTerm`** returns a String that contributes to the target function of an optimization problem in MiniZinc, by weighing adherence positively or breakage negatively.
- **`getFreeVars`** is called for a local neighborhood search based on the Solution `solution` where the target is to optimize with one specific constraint in mind. The result is a selection of the DIMACS representation of decision variables that are likely to allow an optimization in terms of this specific constraint. When multiple neighborhood functions should be tried out by the optimizer, multiple sets of decision variable identifiers can be returned, as the return type is a Seq[Seq[Int]] and every Seq[Int] will be used for a single local search.

Constraint Domain

Constraints formulate statements about single or several decision variables. But because the decision variables are generated at runtime, we have to find the relevant decision variables also at runtime. The class ConstraintDomain is the mean to that end and shown in listing 4.15.

```
1  class ConstraintDomain(val filter: (DecisionVariable, Map[String, String]) => Boolean, output: Any
       => Unit) {
```

```
2   def getVars(decVariables: Map[Int,DecisionVariable], substitutionEnv: Map[String, String]):
        Map[Int, DecisionVariable] = {
3    decVariables.filter(pair => filter(pair._2, substitutionEnv)).toMap
4   }
5   def getTeachers(decVariables: Map[Int,DecisionVariable], substitutionEnv: Map[String, String]):
        Seq[String] = {
6    getVars(decVariables, substitutionEnv).foldLeft(Seq[String]())((a, b) => a ++
         Seq(b._2.teacher)).distinct
7   }
8   def getClasses(decVariables: Map[Int,DecisionVariable], substitutionEnv: Map[String, String]):
        Seq[String] = {
9    getVars(decVariables, substitutionEnv).foldLeft(Seq[String]())((a, b) => a ++
         b._2.clazz).distinct
10  }
11  def getSubjects(decVariables: Map[Int,DecisionVariable]) : Seq[String] = {
12   getVars(decVariables, Map[String,String]()).foldLeft(Seq[String]())((a,b) => a ++
         Seq(b._2.subject)).distinct
13  }
14 }
```

Listing 4.15: ConstraintDomain Implementation

An object of type ConstraintDomain is created with a filter function, that returns whether a DecisionVariable is *in this domain*, e.g. fulfilling certain qualities and therefore being relevant for a constraint. The filter function also takes a Map[String,String] for static scoping of variables. It can be used in loop constraint, that imposes the same constraint on an iterator variable, that should be assigned to a different entity in each iteration. Generally a DecisionVariable is filtered concerning three qualities: the teacher, the affectedClasses and the subject. For every one of these three properties of a DecisionVariable, there are several productions that create different filter functions. In cases when a union between multiple selectors is required, a new ConstraintDomain is created from the conjunction of all the filter functions of the single ConstraintDomain objects. An example can be seen in listing 4.16:

```
1  case StrategoAppl("Assignment", teacherSelector, classSelector, subjectSelector) => new
       ConstraintDomain((x, env) => ConstraintDomain.from(teacherSelector, output).filter(x, env) &&
       ConstraintDomain.from(classSelector, output).filter(x, env) &&
       ConstraintDomain.from(subjectSelector, output).filter(x, env), output)
2  case StrategoAppl("TeacherIndividual", nameList) => nameList match {
3    case StrategoList(names) =>
4     val list = names.map { x =>
5       x match {
6     case StrategoString(name) => name
7     case _ => ""
8       }
9     }
10    new ConstraintDomain((x, env) => list.map { y => if (env.contains(y)) env(y) else y
          }.contains(x.teacher), output)
11   case StrategoString(name) =>
12    new ConstraintDomain((x, env) => if (env.contains(name)) env(name).equals(x.teacher) else
          name.equals(x.teacher), output)
13 }
```

Listing 4.16: ConstraintDomain Interpretation Example

A ConstraintDomain that results from a production “Assignment” (line 1) filters on all three qualities and we build a filter function from the conjunction of the single ConstraintDomain’s filter functions resulting from the interpretation of `teacherSelector`, `classSelector` and `subjectSelector`. The production “TeacherIndividual” selects either a single or multiple teachers by id. As the id could be bound in the static scope, we have to look it up first.

Constraint Implementations

In this section we introduce the specific constraints that can be imposed onto the solvers so far. Different constraint types can easily be added by implementing the trait Constraint.

Assignment

The simplest constraints are the ones that select one or more decision variables to be true or false. In listing 4.17 it is shown how an Assignment refers to a ConstraintDomain `domain` whose filter functions select the decision variables to be set true or false. The Boolean `assign` determines whether the decision variables shall or shall not be selected.

```
class Assignment(val domain: ConstraintDomain, val assign: Boolean) extends Constraint {
  def apply(problem: Problem, substitutionEnv: Map[String,String], output: Any => Unit) {
    val vars = domain.getVars(problem.decVariables.toMap,substitutionEnv)
    vars.foreach { x => assign match {
      case true => problem.depHelper.setTrue(x._1, "Assign " + x._2.teacher + " to " + x._2.clazz + "
          in " + x._2.subject )
      case false => problem.depHelper.setFalse(x._1, "Forbid Assignment of " + x._2.teacher + " to "
          + x._2.clazz + " in " + x._2.subject )
    }}}
```

Listing 4.17: Assignment Constraint

Imposing such a constraint as hard onto the SAT solver is achieved with a call of `setTrue` or `setFalse` on the DependencyHelper object for each decision variable in the ConstraintDomain object. During feedback interpretation of the solver this constraint could come up as source of a conflict. The constraint is therefore named to allow the interpretation "There is a conflict because you ..." either "Assign teacherX to Gymnasium 5b in english" or "Forbid Assignment of teacherX to Gymnasium 5b in english".

```
def getOptimizationConstraint(problem: Problem, substitutionEnv: Map[String, String]) : String = {
  val vars = domain.getVars(problem.decVariables.toMap, substitutionEnv)
  vars.foldLeft("")((acc, decVar) => acc + "constraint " + decVar._2.toZincVariable() + " = " +
      (assign match {
    case true =>" true;\n"
    case false => " false;\n"}
    ))}
```

Listing 4.18: Assignment Optimization Constraint

In MiniZinc the behaviour as hard constraint is achieved in a similar fashion and we generate the necessary String for each decision variable separately. Listing 4.18 shows how every decision variable `x` will result in a line "`constraint x = true`" (or false respectively).

```
def getTargetFunctionTerm(problem: Problem, substitutionEnv: Map[String,String], weight: Int) :
    String = {
  val vars = domain.getVars(problem.decVariables.toMap, substitutionEnv)
  vars.foldLeft("")((acc, decVar) => acc + (assign match {
    case true =>" - "
    case false => " + "}) + weight + " * " + decVar._2.toZincVariable())
}
```

Listing 4.19: Assignment Target Function Term

When such a constraint is declared as weak, the term expressing the preference for or against an assignment is created by subtraction or addition of the multiplication of the decision variable with the weight this constraint is assigned. A decision variable `x` that should be assigned would result in the term "`- x * weight`" and in "`+ x * weight`" when it should not be selected as shown in listing 4.19. Thereby, the target function value decreases or increases by the weight of this constraint, if any of the decision variables in the ConstraintDomain is selected by the solver.

```
def getFreeVars(solution: Solution, substitutionEnv: Map[String,String]) : Seq[Seq[Int]] = {
  val Assign = assign
  domain.getVars(solution.values, substitutionEnv).foldLeft(Seq[Seq[Int]]())((acc, elem) =>
      solution.values(elem._1).isSet match {
    case Assign => acc
    case _ => acc ++ Seq(elem._2.getSimilarSwapTeacher(solution.values).map { x => x.id })})
}
```

Listing 4.20: Assignment Neighborhoods

We have not conducted further research on comparing metrical selection of decision variables to be released in optimizing runs. So as of now, we simply generate one collection of decision variables to be released that should allow a simple swap between teachers. As the goal of optimizing regarding this specific constraint should be to assign the teacher to the specific workload unit, a simple swap should suffice in this case. So if the desired selection is not found, the function `getSimilarSwapTeacher` returns the possible swap candidates as shown in listing 4.20.

```
def isFulfilled(solution: Solution, substitutionEnv: Map[String,String]) : Boolean = {
  val Assign = assign
  val vars = domain.getVars(solution.values,substitutionEnv)
  vars.foldLeft(true)((acc, elem) => solution.values(elem._1).isSet match {
    case Assign => acc
    case _ => return false
  })}
```

Listing 4.21: Assignment Fulfilled

The constraint is fulfilled by a specific solution, when every decision variable in the ConstraintDomain is selected or not selected according to `assign`. If the selection of one single decision variable in the ConstraintDomain does not correspond to `assign`, there is a possibility for a better solution and optimizing may reveal this better solution and `false` is returned as shown in listing 4.21.

Implication

An Implication constraint establishes IF-THEN relationships between decision variables. It is primarily used for rules like "If teacher X teaches bilingual history in class Gymnasium 6b, then teacher X teaches english in class Gymnasium 6b".

```
case class Implication(val lhs: ConstraintDomain, val rhs: ConstraintDomain) extends Constraint {
  def apply(problem: Problem, substitutionEnv: Map[String, String], output: Any => Unit) {
    lhs.getVars(problem.decVariables.toMap, substitutionEnv).foreach( lh =>
       rhs.getVars(problem.decVariables.toMap, substitutionEnv).keys.foreach { x =>
           problem.depHelper.implication(lh._1).implies(x).named(lhs.toString() + rhs.toString()) }
       )}
```

Listing 4.22: Implication Constraint

The constructor takes two ConstraintDomain objects `lhs` and `rhs`. The decision variables in the ConstraintDomain `lhs` (left-hand side) should imply the decision variables in the ConstraintDomain `rhs` (right-hand side). For lack of a more advanced Boolean interpretation in our current design, we simply establish one-to-one implications between the decision variables in `lhs` and `rhs`. Thus, if any decision variable on the left-hand side is selected by the solver, all decision variables on the right-hand side must be selected as well. Listing 4.22 shows how the constraint is established by calling `implication` with the left-hand-side decision variable, which returns an ImplicationRHS object that completes the implication by setting the right-hand-side decision variable with `implies`. The call of `named` is necessary as all constraints need to be named in order to give information on conflicts later on.

```
def getOptimizationConstraint(problem: Problem, substitutionEnv: Map[String, String]): String = {
  lhs.getVars(problem.decVariables.toMap, substitutionEnv).foldLeft("")( (accLHS, lh) => accLHS +
     rhs.getVars(problem.decVariables.toMap, substitutionEnv).foldLeft("")((accRHS, rh) =>
   accRHS + "constraint " + lh._2.toZincVariable() + " -> " + rh._2.toZincVariable() + ";\n")
  )}
```

Listing 4.23: Implication Optimization Constraint

In MiniZinc an implication can be stated in the form "`constraint lhs -> rhs;`", so we build the String representation of this constraint as depicted in listing 4.23 analogue to the `apply` function.

```
def getTargetFunctionTerm(problem: Problem, substitutionEnv: Map[String, String], weight: Int):
    String = {
  lhs.getVars(problem.decVariables.toMap, substitutionEnv).foldLeft("")( (accLHS, lh) => accLHS +
```

```
    rhs.getVars(problem.decVariables.toMap, substitutionEnv).foldLeft("")((accRHS, rh) =>
  accRHS + " + " + weight + " * (" + rh._2.toZincVariable() + " - " + lh._2.toZincVariable() + ")")
)}
```

Listing 4.24: Implication Target Function Term

The representation as weak constraint can be formulated as "`- weight * lhs * (rhs - lhs)`". So again, for every one-to-one implication we add such a term to the target function representation as shown in listing 4.24. It acts as a penalty term adding the weight of the constraint to the target function in case the `lhs` is selected but the `rhs` is not.

```
def getFreeVars(solution: Solution, substitutionEnv: Map[String, String]): Seq[Seq[Int]] = {
  val rh = rhs.getVars(solution.values, substitutionEnv).values.toSeq
  lhs.getVars(solution.values, substitutionEnv).foldLeft(Seq[Seq[Int]]()) { (result, lh) =>
    result ++ rh.foldLeft(Seq[Seq[Int]]())((acc, id) => acc ++
        Seq(lh._2.getSimilarSwapTeacher(solution.values).map { x => x.id } ++ Seq(lh._1) ++
        id.getSimilarSwapTeacher(solution.values).map { x => x.id }))
}}
```

Listing 4.25: Implication Neighborhoods

In listing 4.25 we reflect that in case a single one-to-one implication is not fulfilled, there are two ways the penalty term in the target function can be influenced to evaluate to zero. First, the `lhs` can be set to `false`. We therefore release it as well as the decision variables of other teachers that might be assigned to the workload unit. Second, the `rhs` can be set to `true` and we again try to establish a simple swap.

WorkloadConstraint

A WorkloadConstraint is a constraint that limits the workload of a specific teacher to be lower equal, greater equal or equal to a specific value.

```
case class WorkloadConstraint(val domain: ConstraintDomain, targetValueGenerator: Int => Int,
    relationMode: Int) extends Constraint {
  def apply(problem: Problem, substitutionEnv: Map[String,String], output: Any => Unit) {
    val originVariables = domain.getVars(problem, substitutionEnv).map { x => x._2.id }.toArray
    val weightedVariables : Seq[WO[Int]]= originVariables.map { x => WeightedObject.newWO[Int](x,
        problem.decVariables(x).weight.toLong) }.toSeq
    val overtimeAcc = domain.getTeachers(problem, substitutionEnv).foldLeft(0.0)((acc, string) =>
        acc + problem.teachers(string).targetWorkload)
    val teachers = domain.getTeachers(problem, substitutionEnv).foldLeft("")((acc, elem) => acc +
        elem + ", ").dropRight(2)
    relationMode match {
      case AggregationConstraint.GREATEREQUAL => problem.depHelper.atLeast("Workload of teachers " +
          teachers + " should be at least " + targetValueGenerator(overtimeAcc.toInt),
          BigInteger.valueOf(targetValueGenerator(overtimeAcc.toInt)), weightedVariables:_*)
      case AggregationConstraint.LOWEREQUAL => problem.depHelper.atMost("Workload of teachers " +
          teachers + " should be at most " + targetValueGenerator(overtimeAcc.toInt),
          BigInteger.valueOf(targetValueGenerator(overtimeAcc.toInt)), weightedVariables:_*)
      case AggregationConstraint.EQUAL =>
        problem.depHelper.atLeast("Workload of teacher " + teachers + " should be at least " +
            targetValueGenerator(overtimeAcc.toInt),
            BigInteger.valueOf(targetValueGenerator(overtimeAcc.toInt)), weightedVariables:_*)
        problem.depHelper.atMost("Workload of teacher " + teachers + " should be at most " +
            targetValueGenerator(overtimeAcc.toInt),
            BigInteger.valueOf(targetValueGenerator(overtimeAcc.toInt)), weightedVariables:_*)
    }}
```

Listing 4.26: WorkloadConstraint

Listing 4.26 shows how a WorkloadConstraint is established by retrieving the teacher from the ConstraintDomain `teacher`. Because the syntactical constructs in our DSL formulate these constraints as relative to the target workload ("Overtime/Workreduction should be ..."), the value that the workload should be limited to is calculated at runtime relative to the target workload of the teacher. The function `targetValueGenerator` is applied to the target workload to retrieve this value. The Int

`relationMode` is matched against the static values that symbolize $\geq$, $\leq$ and == relationships. This constraint is imposed as a Pseudo-Boolean constraint: Every decision variable that reflects an assignment of the teacher has to be weighted with the corresponding workload. The solver uses objects of the class WeightedObject, that encapsulate the decision variable and its weight. The DependencyHelper's functions `atLeast` and `atMost` take a collection of WeightedObject and the cardinality. Because the target workload is not necessarily an integer, we have to implement some inaccuracy concerning these constraints: the target workload is rounded to be an integer.

```
1  def getOptimizationConstraint(problem: Problem, substitutionEnv: Map[String, String]) : String = {
2    val vars = domain.getVars(problem.decisionVariables.toMap, substitutionEnv)
3    val overtimeAcc = domain.getTeachers(problem.decisionVariables.toMap,
         substitutionEnv).foldLeft(0.0)((acc, string) => acc +
         problem.teachers(string).targetWorkload)
4    relationMode match {
5      case AggregationConstraint.GREATEREQUAL => vars.foldLeft("constraint ")((acc, decVar) => acc +
           decVar._2.toZincVariable() + " * " + decVar._2.weight + " + ").dropRight(2) + ">= " +
           (targetValueGenerator(overtimeAcc)) + ";\n"
6      case AggregationConstraint.LOWEREQUAL => vars.foldLeft("constraint ")((acc, decVar) => acc +
           decVar._2.toZincVariable() + " * " + decVar._2.weight + " + ").dropRight(2) + "<= " +
           (targetValueGenerator(overtimeAcc)) + ";\n"
7      case AggregationConstraint.EQUAL => vars.foldLeft("constraint ")((acc, decVar) => acc +
           decVar._2.toZincVariable() + " * " + decVar._2.weight + " +").dropRight(1) + " == " +
           targetValueGenerator(overtimeAcc) + ";\n"
8    }
9  }
```

Listing 4.27: WorkloadConstraint Optimization Constraint

In MiniZinc a WorkloadConstraint can be stated as an inequality or equality where we multiply the decision variables with their weight on the left-hand side and state the target value on the right-hand side as shown in listing 4.27. To reflect this constraint in the target function we have to construct a rather complex term that penalizes any deviation in case of an equality constraint and for greater or equal and lower or equal the penalty has to reflect only one direction of deviation. The adherence or breakage of a constraint should always be evaluated equally, e.g. the penalty term should evaluate to the same value when a teacher's workload is one hour less than it should be and also when it is two hours less than it should be. We therefore need terms that evaluate only to 1 and 0 or to -1 and 0.

$$0^{t mod (a+0^a)} - 1 \tag{4.1}$$

Equation 4.1 evaluates to -1 whenever an actual value a is lower than the target value t and to 0 otherwise.

$$0^{t mod (a+0^a)} \tag{4.2}$$

Equation 4.2 evaluates to 1 whenever an actual value a is greater than the target value t and to 0 otherwise.

$$0^{abs(t-a)} \tag{4.3}$$

Equation 4.3 evalutes to 1 only when an actual value a is equal to the target value t and to 0 otherwise. In listing 4.28 we construct the MiniZinc terms that correspond to these equations. A constraint stating that the workload w should be lower than t for example would be formulated as "`+ weight * pow(0, t mod (w + pow(0,w))`".

```
1  def getTargetFunctionTerm(problem: Problem, substitutionEnv: Map[String, String], weight: Int) :
       String = {
2    val vars = domain.getVars(problem.decisionVariables.toMap, substitutionEnv)
3    val overtimeAcc = domain.getTeachers(problem.decisionVariables.toMap,
         substitutionEnv).foldLeft(0.0)((acc, string) => acc +
         problem.teachers(string).targetWorkload)
4    relationMode match {
```

```
5       case AggregationConstraint.EQUAL => " - " + weight + " * pow(0,abs("+
            targetValueGenerator(overtimeAcc) + " - " + vars.foldLeft("")((acc, elem) => acc +
            elem._2.toZincVariable() + " * " + elem._2.weight + " + ").dropRight(3) + ")" + ")"
6       case AggregationConstraint.GREATEREQUAL => " + " + weight + " * (pow(0,"+
            targetValueGenerator(overtimeAcc).toInt + " mod " + "(pow(0," + vars.foldLeft("")((acc,
            elem) => acc + elem._2.toZincVariable() + " * " + elem._2.weight.toInt + " +
            ").dropRight(3) + ") + " + vars.foldLeft("")((acc, elem) => acc + elem._2.toZincVariable()
            + " * " + elem._2.weight.toInt + " + ").dropRight(3) + "))-1)"
7       case AggregationConstraint.LOWEREQUAL => " + " + weight + " * pow(0,"+
            targetValueGenerator(overtimeAcc).toInt + " mod " + "(pow(0," + vars.foldLeft("")((acc,
            elem) => acc + elem._2.toZincVariable() + " * " + elem._2.weight.toInt + " +
            ").dropRight(3) + ") + " + vars.foldLeft("")((acc, elem) => acc + elem._2.toZincVariable()
            + " * " + elem._2.weight.toInt + " + ").dropRight(3) + "))"
8     }
9   }
```

Listing 4.28: WorkloadConstraint Target Function Term

To optimize in the sense of a WorkloadConstraint we relax all the decision variables that concern an assignment that is included in the ConstraintDomain `domain` and the decision variables that are potential swap candidates. This allows for all possible permutations of assignments to be evaluated as solution candidate as shown in listing 4.29.

```
1   def getFreeVars(solution: Solution, substitutionEnv: Map[String,String]) : Seq[Seq[Int]] = {
2     val vars = domain.getVars(solution.values, substitutionEnv)
3     if (isFulfilled(solution, substitutionEnv))
4       Seq()
5       else
6         Seq(vars.filter(decVar => decVar._2.weight > 0).foldLeft(Seq[Int]())((acc, decVar) => acc ++
              decVar._2.getSimilarSwapTeacher(solution.values).map { x => x.id }))
7   }
```

Listing 4.29: WorkloadConstraint Neighborhoods

AggregationConstraint

The implementation of AggregationConstraint is equal to the one of WorkloadConstraint except for:

1. The decision variables are not weighted, so only the number of assignments respectively selected decision variables is constrained.

2. No inaccuracy occurs, since decision variables are either 1 or 0 in MiniZinc.

ForEachTeacher

A ForEachTeacher constraint allows to iterate over a set of teachers referred to by the ConstraintDomain `teachers`. The teachers referred to by the ConstraintDomain `excludedTeachers` will not be considered in the iteration. The String `substitute` denotes the identifier that will be bound to the respective teacher in each iteration step. The Seq[Constraint] `constraints` contains all the Constraint that should be imposed in every iteration step.

```
1   case class ForEachTeacher(val teachers: ConstraintDomain, val excludedTeachers: ConstraintDomain,
        val substitute: String, val constraints: Seq[Constraint]) extends Constraint
2     def apply(problem: Problem, substitutionEnv: Map[String,String], output: Any => Unit) {
3      teachers.getTeachers(problem.decisionVariables.toMap, substitutionEnv).filter { x =>
           !excludedTeachers.getTeachers(problem.decisionVariables.toMap, substitutionEnv).contains(x)
           }.foreach { x => constraint.foreach( c => try { c.apply(problem, substitutionEnv +
           (substitute -> x), output)} catch {
4           case e: org.sat4j.specs.ContradictionException => output("Not solvable: " + x)
5           throw e
6         }) }
```

```
7   }
```

Listing 4.30: ForEachTeacher Constraint

To impose the constraint as hard onto the PB solver the set of teachers is retrieved. Listing 4.30 shows how for every teacher in the set we build a static scope that binds `substitute` to the identifier of the teacher. For every Constraint in `constraints` the function `apply` is called by passing `problem` and the modified static scope. The functions concerning the formulation of the optimization problem are implemented analoguously. The single String representations or target function terms of the constraints are simply concatenated as shown in listing 4.31:

```
1    def getOptimizationConstraint(problem: Problem, substitutionEnv: Map[String, String]) : String = {
2     teachers.getTeachers(problem.decisionVariables.toMap, substitutionEnv).filter { x =>
          !limitingScope.getTeachers(problem.decisionVariables.toMap, substitutionEnv).contains(x)
          }.foldLeft("")((acc, teacher) => acc + constraint.foldLeft("")((ac, cons) => ac +
          cons.getOptimizationConstraint(problem, substitutionEnv + (substitute -> teacher))))
3   }
4   def getTargetFunctionTerm(problem: Problem, substitutionEnv: Map[String,String], weight: Int) :
        String = {
5    teachers.getTeachers(problem.decisionVariables.toMap, substitutionEnv).filter { x =>
          !limitingScope.getTeachers(problem.decisionVariables.toMap, substitutionEnv).contains(x)
          }.foldLeft("")((acc, teacher) => acc + constraint.foldLeft("")((ac, cons) => ac +
          cons.getTargetFunctionTerm(problem, substitutionEnv + (substitute -> teacher),weight)))
6   }
7   def getFreeVars(solution: Solution, substitutionEnv: Map[String,String]) : Seq[Seq[Int]] = {
8    teachers.getTeachers(solution.values, substitutionEnv).filter { x =>
          !limitingScope.getTeachers(solution.values, substitutionEnv).contains(x)
          }.foldLeft(Seq[Seq[Int]]())((acc, teacher) => acc ++
          constraint.foldLeft(Seq[Seq[Int]]())((acc1,const) => acc1 ++ const.getFreeVars(solution,
          substitutionEnv + (substitute -> teacher))))
9   }
```

Listing 4.31: ForEachTeacher Optimization Implementation

ForEachClass

The meaning and implementation for the ForEachClass constraint is equal to the one of ForEachTeacher, except that the iteration steps bind `substitute` to the class identifiers.

Deputations

Deputations are virtual workload units that are assigned to teachers who are assigned to a specific amount of specific workload units. Its implementation is shown in listing 4.32.

```
1  class Deputation(val name: String, val teacherIterator: String, val constraints: Seq[Constraint]) {
2   def appliesToTeacher(teacher: String, solution: Solution) : Boolean = {
3    constraints.foldLeft(true)((acc, elem) => elem.isFulfilled(solution,
         Map[String,String](teacherIterator -> teacher)) match {
4     case true => acc
5     case false => return false
6    })}}
```

Listing 4.32: Deputation Implementation

A Deputation has a `name` and is assigned to a teacher, when the Constraint `constraint` holds. When a Deputation applies to a teacher, the planner recalculates the target workload of the teacher. Usually an applied Deputation should lower the target workload of a teacher to leave less worklod capacity for other assignments. For now there is no mechanism to easily introduce the concept of a Deputation to either the CSP nor the optimization problem. This is due to the conditional behaviour that would be needed. Sat4j offers logical gates that could consider that an additional constraint should be imposed

when certain other constraints are fulfilled or some decision variables are selected. But this is only available in the native SAT solvers of the library and not for the PB solvers. In case logical gates would be provided for PB solvers, every WorkloadConstraint for a specific teacher would have to be replicated with the recalculated limits. MiniZinc allows conditional value assignments via an if-then-else construct. But the Boolean expression that is evaluated in order to select either the then- or the else-branch must not contain decision variables. Therefore deputations are checked against solutions found in our solving process and are then assigned to the relevant teachers, as we depict in chapter 5.

5 Solving Process Design

In this chapter we show the solving process of the DSL backend. An overview of the single steps is given below:

1. Find inconsistencies in problem declaration (PB solver).
2. Find an efficient solution (PB solver).
3. Generate and optimize subproblems (MiniZinc).

Figure 5.1 shows the process steps. In section 5.1 we depict how the first run of the PB solver is arranged and how its run influences the further process (step 1). In section 5.2 the iterative SAT solving with feedback interpretation is explained (step 2). Section 5.3 depicts the setup of local searches to further optimize on basis of the solution found in the previous steps (step 3).

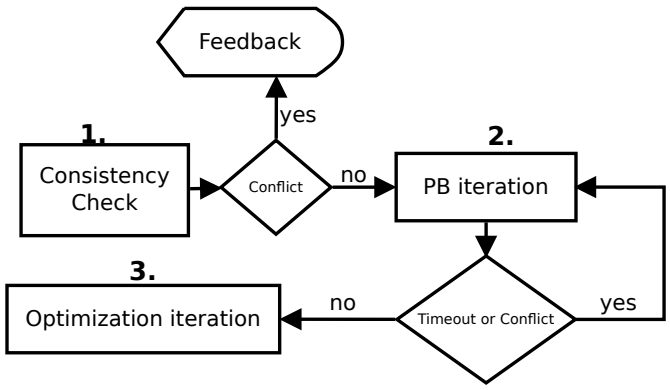

Figure 5.1.: Solving Process

5.1 Concistency Check

As we have identified inference to be a strength of SAT solving and our PB solver makes use of this native strength, we exploit these qualities in order to find inconsistencies in the user's problem declaration. A user for example might declare a constraint that forbids a specific teacher to teach a specific subject and limit the overtime of all teachers. Then it might occur, that the capacity of the remaining teachers with this subject does not suffice. Such an inconsistency shall and will be found in the concistency check. Hence, we simply call the function `hasASolution` on the DependencyHelper object, which at this stage only considers the constraints declared by the user.

Feedback interpretation

A run of the SAT solver will try to find a solution until one is found, unsatisfiability has been proven or the timeout has expired.

Satisfiable
When a solution has been found we go to the next step and try to find a more efficient solution.

Unsatisfiable

In case of unsatisfiability the DependencyHelper returns the conflicting constraints. We simply output the list of the conflicting constraints for the user to interpret. An example for such a situation is shown in listing 5.1.

```
1 Assign teacher X to libraryservice
2 Workload of teacher X should be at most 1
```

Listing 5.1: Feedback: Workload Conflict

Here the derived constraint that was generated from the presetting of teacher X to the special assignment "libraryservice" results in a workload of more than one hour. The user now has to either remove the assignment of teacher X to "libraryservice" or adapt the constraintness of her workload to allow at least the workload that is caused by an assignment of "libraryservice".

Timeout

In case of a timeout we have to increase the timeout value, as we are currently facing the representation of the problem that only includes the necessary constraints and not the additional ones we will impose in section 5.2. A discussion of performance will be found in chapter 6 to give a picture of problem sizes and the resulting performance.

5.2 Efficiency iteration

When a consistent problem specification has been proven in the consistency check, we try to find the most efficient lower bound solution. Efficiency is expressed by a deliberate measure: In this thesis efficiency means the least deviation from the target workload for each teacher. We interprete efficiency as a non-discriminatory measure and evaluate every deviation from the target workload of every teacher equally. Other possible efficiency measures could also be realized: As few teachers with deviation as possible (non-discriminatory), high/low diversity in terms of teachers per class or discriminatory version that limit these efficiency measures on specific set of teachers.

Efficiency Constraint Generation

To achieve an efficient lower bound solution, we impose additional constraints onto the PB solver. In case of the chosen efficiency measure we calculate the most strict bounds by equally distributing the difference of needed workload and target workload between all teachers. In case of 100 hours of overcapacity, e.g. needed workload equals 200 hours and the target workload is 300 hours for ten teachers, at least ten hours of work reduction result from the optimal solution. In this case the optimal overtime would be 0. With this calculation we generate additional hard constraints for the user declared problem specification for every teacher as depicted in listing 5.2.

```
1   new WorkloadConstraint(teacherScope, x => x + upperBound, AggregationConstraint.LOWEREQUAL)
2   new WorkloadConstraint(teacherScope, x => x - lowerBound, AggregationConstraint.GREATEREQUAL)
```

Listing 5.2: Efficiency Constraint Generation

If the problem specification already contains a hard constraint on the workload of a teacher, we do not generate those constraints. In case of an equality constraint the preset value might not be in the range given by the lower and upper bound. In case of a user defined inequality for the workload of the teacher, they are either stricter than the newly generated ones and the generated ones would do no harm aside from redundancy. Or they are less strict and the newly generated one may result in more efficiency for this teacher. But there could also be a conflict in case the user declares that the overtime of a teacher should be greater or equal to a specific value that lies outside of the range defined by the current lower and upper bound.

As in section 5.1 we have the three possible outcomes. If a solution is found, we can stop and continue with local optimizing which is described in section 5.3. In case of a timeout we generally have two options: to increase the timeout value and try again or to relax the constraints and try again. For now the user interaction with the solving process is limited to the iterative specification, e.g. once a problem specification is compiled, the user can't influence the process aside from its termination. We therefore did not implement an automatic timeout increase, as an appropriate value depends too much on the user's hardware.

Unsatisfiable

In case the solver proves unsatisfiability we look at the collection of conflicting constraints. With the knowledge from the steps in section 5.1, that there is a possible solution given the current problem specification, and that there are only new bound constraints on the workload of possibly all teachers, the interpretation is rather simple. That is because there are two ways the newly added constraints can lead to unsatisfiability:

1. The workload of the teachers of a specific subject is constrained in the way that the allowed workload sum of all those teachers doesn't suffice for the required lessons.

2. The actual needed lessons in some subjects do not amount to a value that allows a distribution among teachers so that all lower bound workload constraints can be fulfilled.

In the first case we search for a WorkloadConstraint that is a LOWEREQUAL constraint, meaning limiting the workload with an upper bound, in the collection of conflicting constraints. When such a constraint is found, we increase the upper bound and try to solve again. In listing 5.3 the output of such a situation is shown. The "Require at least..." constraints show that there are too many `german` lessons, e.g. a satisfaction could be achieved by removing the `german` lessons or lowering the resulting workload of `german` lessons. The "Workload of teacher..." constraints signal that the listed teachers could be the key to achieve satisfiability by increasing their individual upper bound or their target workload.

```
1 Require at least one teacher for class haupt5a in subject german
2 Require at least one teacher for class haupt5b in subject german
3 Require at least one teacher for class haupt5c in subject german
4 Workload of teacher L should be at most 21 hours
5 Workload of teacher M should be at most 21 hours
6 Workload of teacher N should be at most 21 hours
```

Listing 5.3: Feedback: Too Many German Lessons

In the second case we search for a WorkloadConstraint that is a GREATEREQUAL constraint limiting the workload with a lower bound. Respectively the lower bound is decreased and a new run is started. In listing 5.4 the output of such a situation is shown. The "Require at most..." constraints show that there are too few `english` lessons, i.e. a satisfaction could be achieved by adding `english` lessons or increasing the resulting workload of `english` lessons. The "Workload of teacher..." constraints signal that the listed teachers could be the key to achieve satisfiability by decreasing their individual lower bound or their target workload.

```
1 Require at most one teacher for class haupt5a in subject english
2 Require at most one teacher for class haupt5b in subject english
3 Require at most one teacher for class haupt5c in subject english
4 Workload of teacher X should be at least 17
5 Workload of teacher Y should be at least 26
6 Workload of teacher Z should be at least 26
```

Listing 5.4: Feedback: Not Enough English Lessons

In every iteration of PB solving exactly one set of constraints leading to a conflict is shown. A new iteration might bring the same set of constraints as the bounds have not been relaxed enough. If a new set of constraints is shown, the formerly shown set of constraints is not leading to a conflict anymore. The new set of conflicting constraints is a result from relaxation of the bounds. In terms of efficiency this set of constraints has to be regarded as causing more inefficiency compared to the ones from the iterations before. Therefore the iterative approach appears very reasonable:

- More iterations yield more insights about the weak points of the problem declaration.
- The last iteration before a solution is found shows the constraints that should be "treated" first in order to achieve more efficiency.

This iterative approach might also be used for further exploration. The iteration we depicted in this section searches for a minimum of total deviation from target workload. Another efficiency measure could be the total amount of teachers that have a deviation of target workload at all. In this case we would state WorkloadConstraints of type EQUAL and relax them according to the solver feedback. However, this lies outside of the scope of this thesis.
At some point the iterative approach should yield a solution that is at least as efficient as the first solution we found in section 5.1. We exploited the performance capabilities of the PB solver and now proceed with the more flexible optimizer.

5.3 Optimization with Local Search

So far only hard constraints have been considered in the solving process. This is due to the limited capabilities of the PB solver in terms of stating complex target function terms but primarily because we use its performance advantage to fastly retrieve a lower bound solution. We now use the optimizer's expressiveness to also apply weak constraints by stating a target function and by alteration of the current solution we found in the step before.

Problem formulation

To formulate an optimizing problem we need the decision variables, the workload weights that correspond to the decision variables, hard constraints and a target function that evaluates a solution. Stating the optimization problems relative to the solution we found with the PB solver, we also need to relax a set of decision variables in order for the optimizer to explore different solution candidates.

Workload weights

During PB solving the weights were added everytime a Pseudo-Boolean constraint was added. For debugging purposes we declare them centrally as float variables like shown in listing 5.5:

```
1 float: Creal6bSdeutsch0 = 5.0;
```

Listing 5.5: MiniZinc Weight Declaration

Translation of decision variables

The decision variables already have been generated for PB solving and can simply be translated into decision variables for the optimizer. With MiniZinc we are not bound to a specific format like DIMACS for the PB solver and can choose unique identifiers freely. Especially for debugging purposes we found it most reasonable to name decision variables according to their properties and chose the format that is implemented in the function *toZincVariable*. If we add them to the optimizer model as decision variable an example would look like shown in listing 5.6:

```
1 var bool: TschmidtCgym5cSenglish0 :: output_var;
```

Listing 5.6: MiniZinc Decision Variable

It declares the decision variable `TschmidtCgym5cSenglish0 ∈ [true,false]`. The annotation "`:: output_var`" instructs the optimizer to print the chosen value for this variable in the found solutions.
This translation of a decision variable again creates the problem in its initial state, leaving the whole search space for exploration. But the greatest part of decision variables should be fixed in order to explore only a small neighborhood with a local search. We therefore state only the decision variables returned by the neighborhood functions of the weak constraint as MiniZinc variable. The rest remains fixed according to the solution from the efficiency iteration. The decision variable from listing 5.6 translated as a fixed parameter is shown as selected (line 1) and unselected (line 3) in listing 5.7:

```
1 bool: TschmidtCgym5cSenglish0 = true;
2 ----------------------------------
3 bool: TschmidtCgym5cSenglish0 = false;
```

Listing 5.7: MiniZinc Fixed Variable

Hard constraints

The model's underlying hard constraints that require a workload unit to be assigned to a specific number of teachers are added as hard constraints in the MiniZinc model in an equal way. Listing 5.8 shows the constraint that states that class 5a needs one teacher in its `english` lesson. Note that an addition of Boolean variables takes place. This is valid due to type coercion in MiniZinc (`true` is coerced to 1 and `false` to 0).

```
1 constraint TxCgym5aSenglish + TyCgym5aSenglish + TzCgym5aSenglish = 1;
```

Listing 5.8: MiniZinc Inherent Constraint

Every hard constraint declared by the user provides its own representation as a hard constraint in a MiniZinc model. A hard constraint that assigns teacher x to the `english` lesson from listing 5.8 would be expressed as shown in listing 5.9:

```
1 constraint TxCgym5aSenglish = true;
```

Listing 5.9: MiniZinc Hard Constraint

This might be redundant, if the decision variable was declared as a Boolean with value `true`. The hard constraints we used for efficiency exploration in the SAT iteration will not be added to the model as hard constraints for we want to allow a deviation from efficiency, if the user declared a preference for a less efficient part of the solution. From now on they are considered weak constraints.

Target function

Weak constraints will be considered in the formulation of the target function. Every weak constraint contributes its representation term to the target function as explained in chapter 4.

Optimization iteration

For every weak constraint we formulate a separate optimization problem instance. The individual instances all share the same target function, the same hard constraints and the same workload weights. They differ in the relaxation of the decision variables, meaning that a different set of the decision variables is relaxed relatively to the most efficient solution from PB solving.
Every instance is written to a file, converted to FlatZinc format and then run with the Gecode solver. The

solver traverses the search space and keeps track of the solutions with the least target function value. The solutions are written into a separate file that we parse after the solver found the optimal solution or the timeout has expired. The possible outcomes are as follows:

- Timeout and no solution found results in an empty file, therefore we parse no solution.
- Timeout and a solution has been found or the optimizer returns and an optimal solution has been found.

In case we can parse a solution we compare the new target function value to the former one. If the new one is lower (we minimize) than the last one we continue the iteration with the new solution. The parsing creates a new best solution, possibly changing selection of the relaxed decision variables relative to the former solution. Relative to this new best solution we relax the decision variables for the next iteration based on the metrics of the next weak constraint.
We may check for the adherence to the single weak constraints and skip otpimization in order to save time. But the optimization for a specific weak constraint does not only optimize in order to fulfill this constraint. Because we use a globally valid target function, we might optimize the solution regarding weak constraints other than the current target weak constraint. This effect can result from the metrical selection of the decision variables to be relaxed. The metric for the current constraint may relax decision variables that would not be relaxed by the metrics of all other weak constraints. These relaxed decision variables may lead to a better solution, that would not have been found if we skipped the iteration step.

6 Case Study

To verify that our DSL and the solver backend meet the requirements we got from executives of a big German school, we used a concrete problem instance as reference throughout the development. The usual manual planning process is stretched over weeks once a year. This is either due to changes in the staff or due to the iterational budget-planning from the governmental institution. But also when teachers suddenly fall ill, her assignments have to be redistributed. Therefore we tested whether our prototype is performant enough to be used repeatedly to test different constraints or as a reaction to different problem data. In section 6.1 we outline the extent of the problem instance and in section 6.2 we show how hard and weak constraints influence the runtime. Section 6.3 shows the performance results for a realistic problem specification.

6.1 Problem Data

We modelled the problem instance of the second half of the 2015/2016 school year. The translation of the planning material we got was difficult because of its quantity and its quality. In an Excel-Sheet we could see a list of teachers with their target workload, their career path and subject qualifications that could directly be translated into our DSL. This sheet also included a table of lessonplans and the number of classes. But when trying to model the classes with the given lessonplan, we found that the aggregated workload did not match the info on the sheet. With the second document, that is with the actual assignments of teachers to classes, we could reconstruct the individual workload every class needs. But as the classes could differ greatly from each other, we had to guess what the underlying and most general lessonplan was. Some workload units have subjects that did not reflect a specific subject qualification and teachers with different qualifications were assigned to it. This lead us to implement the different notions of subject declarations depicted in chapter 3.1. For example the subject “Arbeitslehre” is a subject that is only an explicit qualification for teachers in the Haupt-/Realschule career path. But in the actual schedule we found an assignment of a teacher from the Gymnasium career path with subject qualification “German” and “Politik und Wirtschaft”. A possible enhancement of our model could be the feature of the class teacher. For now, we model class teachers by declaring a subject “classteacher” that does not need any qualification. The specification of the reference problem without the teacher specifications can be found in appendix A.1. Its translation resulted in:

- 115 teachers with a total target workload of 2579.68 hours
- 929 workload units with a total amount of 2436.5 hours of workload

This means that there are around 143 hours of overcapacity, target workload that cannot be distributed. With the one teacher that appears in the staff info but is not assigned, probably due to some illness or parental leave, the overcapacity is at 116 hours. Consequently, a solution to this problem will contain at average one hour less than the target workload for each teacher. After preprocessing we found 19420 decision variables for this problem, spanning a search space of possible 2^{19420} solution candidates.

6.2 Hard and Weak Constraints

In our solving process hard constraints are handled by both the PB solver and the optimizer and weak constraints are only considered by the optimizer. We compare the performance of the PB solver with the optimizer by stating a set of constraints as hard in the first run and as weak in the second run. Assuming that the PB solver is more performant than the optimizer, comparison shows the tradeoff between

performance and flexibility. In the first run the user "buys" performance by allowing the situation that no solution will be returned in case the set of hard constraints cannot be satisfied. In the second run the user "pays" for the guarantee of a solution with the time the optimizer needs to search for solutions. Generally this comparison is not able to benchmark the performance of the two tools, as they are not involved in the same way, meaning doing the exact same thing. But the comparison can validate that by arranging the two tools, we picked up the strengths and mitigated the weaknesses of the two tools.
In the first run we declare the set of constraints depicted in listing 6.1:

```
1 Hard Constraint:
2   Workreduction of teacher One = 27
3 Hard Constraint:
4   Workreduction of teacher Two = 0
5 Hard Constraint:
6   Workreduction of teacher Three = 0
7 Hard Constraint:
8   For each teacher X in teachers with qualification german :
9     Overtime of teacher X = 0
```

Listing 6.1: Hard Constraint Set

In the second run we declare the same constraints as weak with priority 4. In the third run we declare all constraints as weak with priority 4 except for the constraint imposed on all the `german` teachers. Note that these are all declarations that result in a WorkloadConstraint. In a WorkloadConstraint the currently implemented relaxation is very generous, i.e. all decision variables of a teacher and corresponding swap candidates are set free. Therefore the optimization runs for each weak constraint in this section loosely represent an upper bound for the runtime performance of a local search with the optimizer. In table 6.1a we present the performance benchmarks for all runs at every checkpoint. In the preprocessing step the first run takes the longest. This is due to the user declared hard constraints that are added to the PB solver in addition to the domain constraints. In run 2 there is only one user declared constraint and in the third run there are two. For the first PB solving step the same reasoning applies. In the first run, where all constraints are declared as hard, the PB solver has to comply with more constraints and takes the longest time. Respectively the second run takes the shortest time. The same applies to the PB iteration. This result is also expected, as it should be harder to find a solution in specific efficient bounds when more additional constraints are issued onto the PB solver.
In the optimizing step the first run only evaluates the solution found during the PB iteration step and runs 788ms. The second run optimizes a total of 20 weak constraints, i.e. two single overtime constraints and 18 resulting from the overtime constraints on all german teachers (which means that there are 18 `german` teachers). In total this optimization iteration takes over six minutes. In the third run only two optimization steps take place and amount to 31 seconds of runtime. In table 6.1b the quality measures

Step	All Hard	All Weak	Combined
Preprocessing	1235	465	729
Consistency check	178	17	63
Efficiency iteration	7491	1795	6061
Optimization	788	413076	31524

(a) Runtime Performance

Measure	All Hard	All Weak	Combined
Target value	144.18	143.18	143.18
Total Overtime	0.5	0	0
Total Reduction	143.68	143.18	143.18
Average deviation	1.92	2.27	1.86

(b) Quality Performance

Table 6.1.: Constraining Approaches

for solutions of these three runs are presented. We see that performance in terms of solution quality is very similar. In run 1 a solution is found that is almost optimal with 144.18 hours of total deviation (143.18 would be optimal). The teachers with deviation (75) have at average 1.92 hours of deviation from their target workload. In run 2 and 3 an optimal solution is found with 143.18 hours of deviation with zero hours of overtime, where the deviation is at average 2.27 hours for 63 teachers in run 2 and 1.86 hours for 77 teachers in run 3.

6.3 Realistic Problem Instance

As section 6.2 has shown, performance of our solving design is performant enough for the incremental approach to test different constraints and to explore the problem. In this section we depict how the solver performs when we try to achieve a solution as closely to the current assignments of the real-life teachers. We therefore considered several characteristics we translated into constraints:

- We preset the assignment as "classteacher" as we found it in the current solution with hard constraints.
- We implement the rule that the "classteacher" should teach as many lessons as possible in the corresponding class. As there are two subjects meaning that a teacher is considered "classteacher", we have to declare a constraint for each of these subjects, as we don't have a possibility to express more complex Boolean expressions.
- We implement the rule that if a class has bilingual lessons, the teacher assigned to "english" lessons should be one of the teachers that teach a bilingual lesson. Again, as there are several bilingual lessons we have to declare a constraint for each of them. And as they directly conflict with each other, they should be weak constraints. Another approach could be to limit the rules to specific grades, i.e. that the implication of "bilingual history implies english" is applied in grades 7 and 8. For the other grades we could impose the implication "bilingual geography implies english". In a setup as such, we could express them also as hard constraints.
- We implement the rule that the teacher that teaches "german" in a class should also teach the lesson "dyslexiasupport". As "dyslexiasupport" is commonly taught as a common course for several classes, we have to declare it as a weak constraint, since not all German teachers of the concerned classes can be assigned to that lesson, for only one teacher is required.
- We implement a deputation that is assigned to a teacher when she is teaching seven hours or more in grades 12 and 13. The deputation will be taken into account after a solution has been found, and can be considered in addition to the quality measures of the solution.

As we declare several rules that affect the majority of teachers and classes as hard constraints, the total number of constraints imposed onto the PB solver will be very high, e.g. every teacher could be assigned as class teacher to almost all classes and for every other possible assignment in one class we will impose a one-to-one implication. Also the total number of optimization runs will be very high, due to the rules stated as weak constraints. The anonymized set of constraints can be found in appendix A.2.

Measure	Realistic before optimization	Realistic after optimization	Actual Assignments 2015/16
Target value	297.68	272	-
Total Overtime	2.25	3.75	37.33
Total Reduction	145.43	146.93	181.01
Average deviation	1.7	1.79	1.52

Table 6.2.: Solution Quality Compared to Actual Assignments

Table 6.2 summarizes the results for the realistic problem declaration. It shows again, that the solution without optimization is a bit less efficient in terms of total deviation from target workload. But from the target value that is produced by applying the target function created for MiniZinc on both solutions, we see that a bit of efficiency has been given up to minimize the target function by fulfilling constraints. The deputation was assigned 15 times, resulting in additional workload, that to one part caused more overtime (5.5 hours) and partly eliminated work reduction (8.5 hours). Due to its extent we did not calculate the target function value for the actual assignments. Therefore we can only argue that either

there are other constraints we were not communicated that resulted in allowing more inefficiency (37.33 hours of overtime) or that our solving process achieves a better solution in terms of efficiency and quality.

6.4 Summary

Figure 6.1 illustrates and emphasizes the results for runtime performance. We can see that with many constraints, especially hard constraints, the runtime increases to over 25 minutes. This is not a fast response time. But considering that many preferences can be expressed with the constraint language and that the efficiency of the solution seems to be better than the current one, it seems adequate compared to a manual process that takes at least as much time. In conclusion this case study supports our design

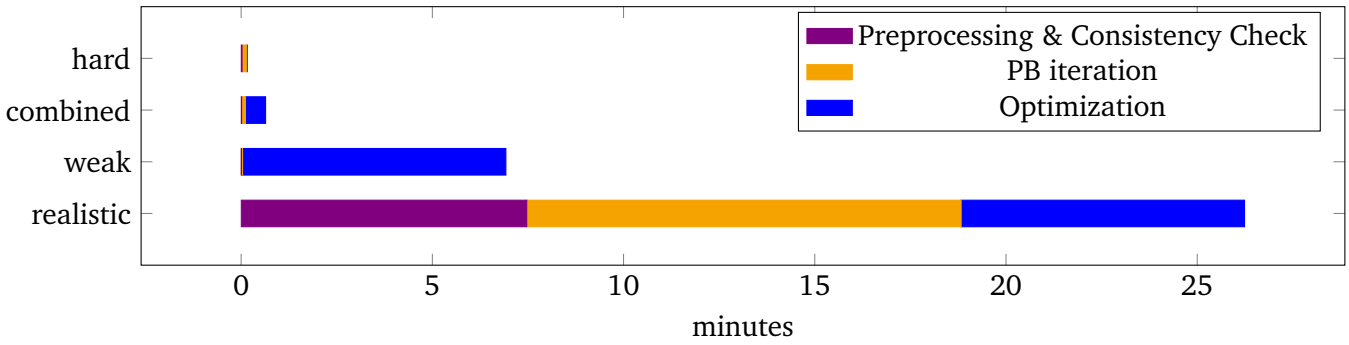

Figure 6.1.: Runtime Performance

choices, as we have designed a more or less performant solving process that yields solutions of a good quality. We summarize the main insights from the comparison of different problem declarations:

- Global optimizing is costly: The implemented neighborhood functions for a WorkloadConstraint release sometimes thousands of decision variables. Then a single optimization run can take several minutes. A local search with 100 decision variables takes less than a second. This further hints at the implementation of local search heuristics that release fewer decision variables in more runs.
- Transforming weak constraints into hard constraints and vice-versa is a good measure to test out different behaviour of the solver in terms of solution quality and runtime performance.

7 User Evaluation

The main goal of this thesis is only achieved, when our DSL is usable by the planners at schools for whom it was designed. We suspect our language to be highly understandable as a result of using syntax that is of course domain specific, referring to terminology known to the user. Secondly, the constraint syntax is very close to statements a user could have made verbally about the specific problem. To test this assumption we conducted a survey with teachers from different schools in order to see how well they understood the concepts of our DSL and the underlying solving process. The complete survey can be found in appendix A.3. We identified five categories that we wanted to test:

1. Workload unit declarations: In order to enter the problem data, the planner needs to understand the formulations used to declare a lessonplan, classes and courses.

2. Teacher qualifications: The qualifications of a teacher directly determine the creation of decision variables. With questions asking for the possibility of an assignment of a teacher to a specific workload unit, we verify that the user understands the search space.

3. Constraint interpretation: In order to express preferences in solutions, a user must understand what different constraints mean and that weak constraints may overrule one another.

4. Rule interpretation: Single constraints referring to single teacher and class entities are probably easier to understand than the ForEachTeacher and ForEachClass constraints. We test the understanding of the extent that such a constraint has.

5. Constraint adherence interpretation: In an incremental planning process with our DSL solutions may reveal that contain unwanted characteristics. Then a planner should impose new constraints forbidding characteristics as such. Therefore we want to test, whether teachers understand if constraints are fulfilled or not.

In the survey we presented small excerpts from complete and compilable problem declarations. Note that we translated our DSL to German in order for the syntax to really be in a domain language for German teachers. In total 15 teachers participated. Seven teachers answered that they had already been engaged in the planning process of teacher assignments. Five teachers declared they had programming experience (primarily in Pascal and Basic). A descriptive statistic for our sample is shown in table 7.1a. We calculated the total number of misinterpretations for each participant and put them into relation with the demographic variables age, programming experience and teacher assignment planning experience. Regression revealed only weak statistical significance on the 0.1 (*) level for programming experience on the total number of misinterpretations as shown in table 7.1b:

Variable	Obs.	Mean	Std. Dev.	Min	Max
# Misinterpretations	15	4.47	3.15926	1	12
Programming Exp.	15	0.33	.48795	0	1
Planning Exp.	15	0.47	.516398	0	1
Age	15	43.47	10.18	25	59

(a) Survey Sample Description

Estimate	Std. Error	t value	Pr(>|t|)
5.95261	3.47401	1.713	0.1146
-3.06816	1.68702	-1.819	0.0963 *
2.26867	1.60038	1.418	0.1840
-0.03501	0.08240	-0.425	0.6791

(b) Regression Results

Table 7.1.: Survey Sample Description & Regression Results

In the following sections we show the performance of the participants in each question category in order to identify structural deficiency of our DSL in terms of comprehensibility.

Workload Unit Declarations

The first five survey questions ask about the resulting workload units of a problem declaration. Figure 7.1 summarizes the overall scores of the participants. It shows that the majority of participants answered

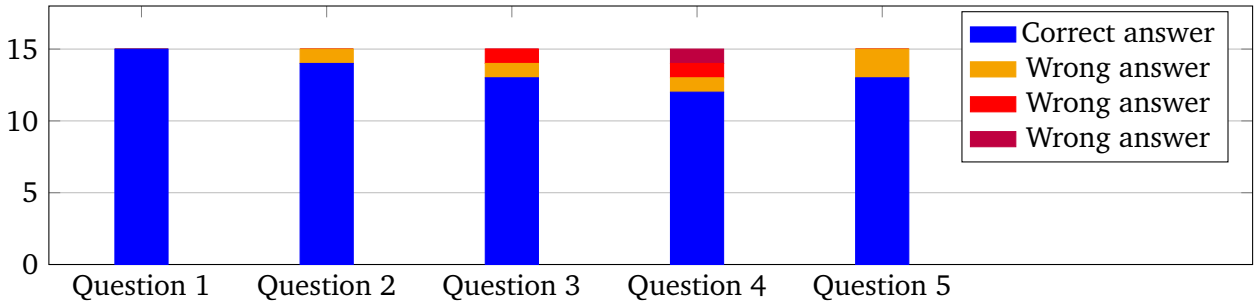

Figure 7.1.: Workload Unit Declaration Interpretation

correctly for the different concepts of workload declaration. All participants recognized the declaration of lessons in a lessonplan correctly (Question 1). Only one participant did not link the class declaration to its corresponding lessonplan, answering 0 hours for a subject that is declared with 2 hours in the lessonplan (Question 2). In Question 3 we presented a class declaration with a deviation from its lessonplan. Again one participant answered 0 hours for a subject declared with 2 hours in the lessonplan and thus did not recognize the link of a class declaration to the corresponding lessonplan. The other wrong answer did not recognize the deviation and answered 2 hours instead of 4. In question 4 we test whether the concept of a common course is understood and asked for the total hours of a subject that is only referenced in such a common course. 13 participants answered correctly (2 hours) and 1 participant answered 0 hours, apparently only looking up lesson declarations in the lessonplan. Another answer was 6 hours. We anticipated that the declaration of 2 hours for 3 classes might be interpreted as 3 courses with 2 hours each. In question 5 we asked for the total amount of German lessons (15 hours) and two participants answered incorrectly 20 hours.

Conclusion

In total the declaration of workload seems to be comprehensive. To expose the link between class declarations and the corresponding lessonplans we suspect two measures to possibly increase comprehensibility. First, we might explicitly reference the lessonplan by an identifier. Secondly, the user could be assisted by the IDE, showing the internally referenced lessonplan.

Teacher Qualifications

The feature that influences the creation of decision variables is teacher qualification. To test whether a non-programmer understands the syntax of our DSL we presented different teacher and class declarations and asked whether certain assignments of teachers to classes would be possible based on subject and school path qualifications. Figure 7.2 summarizes the answers.

In question 6 we asked which subjects an individual teacher may teach. All participants selected correctly “English” and “Mathematics”, showing that the user correctly link this teacher to her subject qualifications. With question 7 we tested the concept that there are different levels of subject qualifications and that teachers on the Haupt-/Realschule level are not allowed to teach on Gymnasium level. Eleven participants recognized this concept correctly and selected only the subjects taught by the teachers with qualifications on the Gymnasium level, when asked for all subjects that could be taught on the Gymnasium level. The four wrong answers included all listed subjects, suggesting that the concept was not clear. In contrast for the individual teacher with only Haupt-/Realschule qualifications all partici-

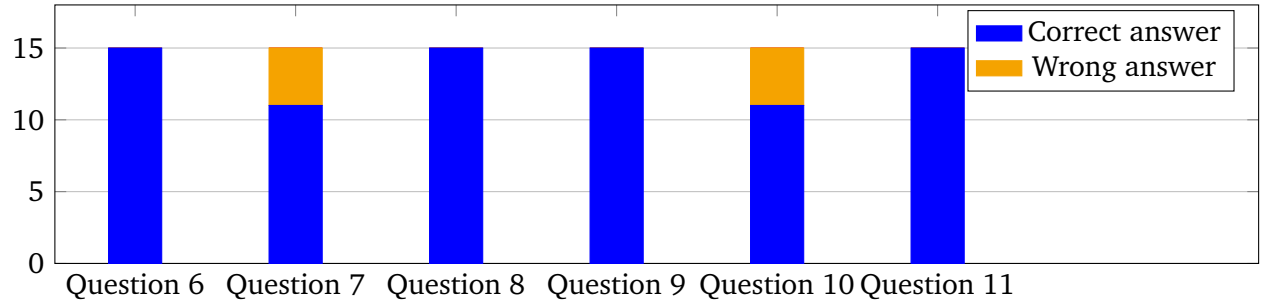

Figure 7.2.: Teacher Qualification Interpretation

pants correctly answered that she could not teach a class on Gymnasium level in question 9. And in question 8 and 11 all participants correctly answered that an individual teacher with Gymnasium level qualifications may teach a class on Haupt-/Realschule level. Four participants answered that an individual teacher with no subject qualifications in German may teach a class in German. We suspect that at this point the participants were scanning only for the qualification levels and missed out that the specified teacher cannot teach German.

Conclusion

In total the concept of teacher qualifications seems to be adequatly expressed in the syntax of our DSL. The wrong answers suggest rather careless mistakes than incomprehensibility.

Constraint interpretation

The constraint language was designed to closely resemble statements that could be made verbally. We tested the comprehensibility of different constraints by asking for their impact on possible solutions. The results are summarized in figure 7.3.

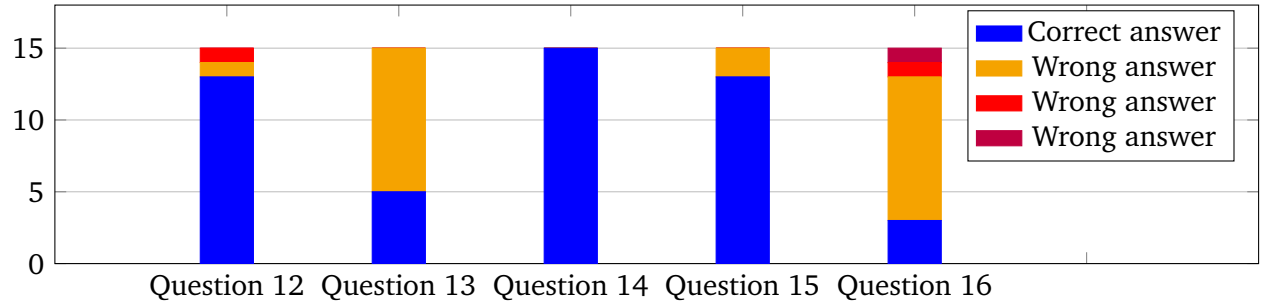

Figure 7.3.: Constraint Interpretation

With question 12 and 13 we tested the comprehensibility of overtime and work reduction constraints. The presented constraints limit the possible workload of the declared teacher to the interval $[23, 27]$. When asked for the minimum workload of the declared teacher, 13 participants answered correctly, one participant either mistook the relation <= for < or simply miscalculated. One participant answered 26 hours, suggesting that the use of the German term “Pflichtstunden”, which was used by the school executives we conferred with, may be misleading. This is due to its meaning being rather obligatory than desirable. When asked for the maximum workload, only five participants answered correctly 27 hours. This is unexpected due to the positive results from question 12. There a mix-up between “<=” and “<” occurred only once.

We established three different constraints to influence a teacher’s possible workload. With overtime and workreduction constraints we refer to the deviation of the target workload in two directions. In

question 14 we wanted to test whether a user might interpret an overtime constraint setting overtime to 0 as only forbidding overtime or whether it was interpreted as a constraint setting the workload to exactly the target workload. All participants answered in favour of our implementation that requires the actual workload to equal the target workload of the teacher. 13 participants correctly identified a hard constraint to forbid any assignment of a teacher to courses in a specific subject in question 15. As we did not suggest to look for a constraint in the declaration, the two wrong answers might result from only looking up the subject qualifications of the teacher. The comprehensibility of a constraint requiring at least two assignments of a teacher in a specific subject was correctly interpreted by three participants. Question 16 asked for the number of classes the specific teacher teaches at minimum in the given subject. Since we presented no classes, the distinction of classes from courses is critical. Semantically the constraint requires at least two assignments. But as one class could have two courses in the specific subject, the correct answer would technically be 1 class. As this turned out to be kind of a trick question, we interpret the 2 classes answer (ten participants) not as a result of incomprehensibility. The answers with 10 and 50 classes seem rather arbitrary with no hint at what caused this misinterpretation.

Conclusion

Except for the poor score in question 13 and the rather complex question 16 the impacts of constraints seem to be comprehensible.

Rule interpretation

An important feature of the constraint language part of our DSL is the ability to express rules that refer to sets of teachers and/or classes. We tested whether the A constraint that should be imposed on all teachers

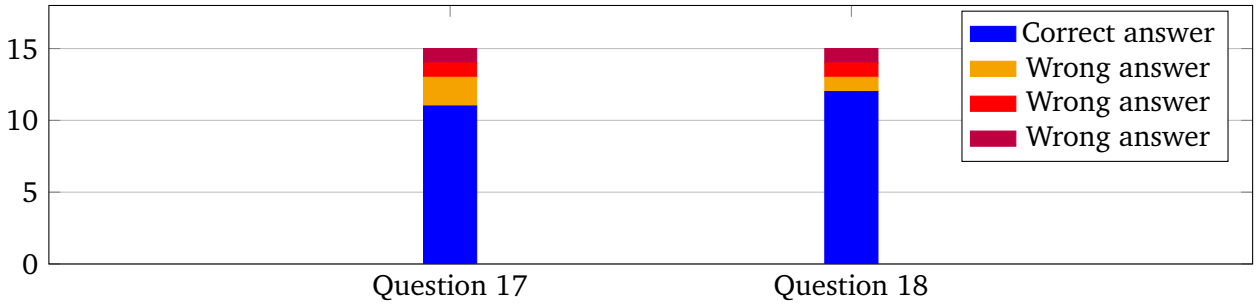

Figure 7.4.: Rule Interpretation

was correctly recognized as being imposed on all teachers by eleven participants. Twelve participants correctly recognized the constraint referred to in question 18 as being imposed on all teachers with a subject qualification in English.

Constraint adherence interpretation

Stating constraints may result in a conflict that will be detected during compilation of the problem declaration. When weak constraints are in conflict, the conflict will not explicitly be stated. Therefore it seems important to test whether the concept of weighting weak constraints in order to determine precedence of them. Figure 7.5 summarizes the results.

When asked which weak constraint would have precedence over the other, all participants correctly selected the constraint with higher weight in question 19. When asking in question 20 for the consequences of those conflicting constraints in a prospective solution, e.g. the assignment of a teacher to a specific English lesson, only nine participants chose the correct teacher. Five participants chose the teacher that would have been assigned if it was not for the precedent constraint. One participant chose a

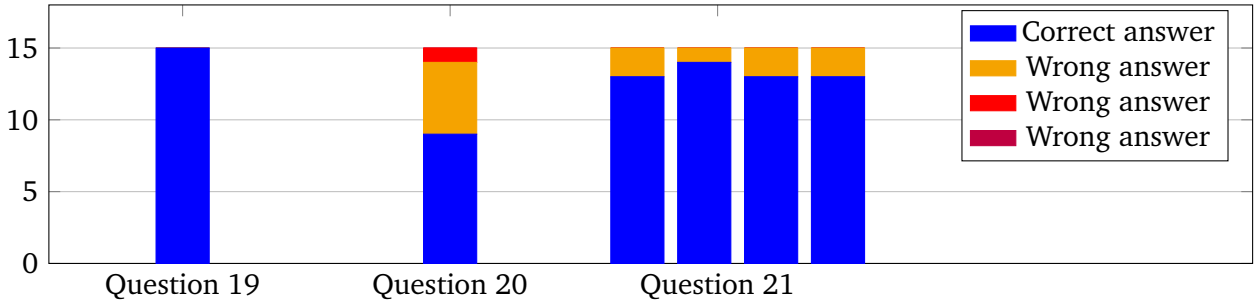

Figure 7.5.: Constraint Adherence Interpretation

teacher without a subject qualification in English. In question 21 we presented four different constraints and parts of a solution containing the aggregated actual workload of the declared teachers. The majority of participants identified them correctly as being fulfilled or not fulfilled.

8 Related and Future Work

Our work seems to be unique in the way we approach a very common and widely researched problem. The majority of literature concerning the teacher assignment problem deals with different solving approaches in the algorithmic sense (e.g. [3], [23] and [22]). On the other hand DSLs for constraint programming are rather solver specific languages and seem to be developed in order to facilitate constraint programming for people familiar with general programming. We briefly present an incomplete list of scientific work that relates to this thesis.

Timetabling

After a solution has been found for the problem depicted in this thesis the planner is confronted with the next NP-hard problem: Distribution of the single workload units over time in an optimal way. Optimality in this case e.g. refers to as few free time in between assignments for teachers as possible or whole free days for some teachers. This problem also has been researched in a vast amount of varieties (for an overview see [4] and [5]), but in this case there are also efficient and accepted commercial solutions, e.g. Untis [1]. Often treated separately from the assignment problem, Gunawan, Ng, and Poh propose a hybrid algorithm to address both problems at the same time.

s-COMMA

With s-COMMA Chenouard, Granvilliers, and Soto introduce a model-driven-development of constraint programming programs. They propose a meta-model to reflect the different aspects of constraint programming such as decision variables, domains, constraints and target functions. Instances of this meta-model can then be created with a graphical modelling tool [6]. The problem domain of teacher assignments could be modelled as such a model instance. As in this thesis, they argue that the implicit declaration of a problem domain that most constraint programmers use, is difficult for beginners. Eventhough we approach a single problem domain and facilitate the creation of a computer-solvable problem with a DSL for this single problem domain, we see interesting links between the two concepts. First, the explicit modelling of the problem domain targets the abstraction from decision variables and constraints. Second, this abstraction seems promising in combination with a local search approach. This is due to the important properties we identified in a constraint to allow local search: Is it fulfilled and how can it probably be fulfilled (neighborhood function)? Combining the quick creation of model instances reflecting specific problem domains and a flexible constraint language based on different constraint types could be explored.

DSLs for Constraint Programming

As we implemented the DSL interpreter and the solving process in Scala, we found Scarab [21] and the Scala embedded DSL for constraint programming Copris[2]. The possibility to express arithmetic constraints and have them automatically translated to the Sat4j solvers is a very powerful and convenient feature, that could improve our own solving process. We did not make use of it, since its compatibility with the current Scala version is not given and maybe hints at a discontinuation.

Teacher Quality

A topic that was not explicitly dealt with in this thesis is teacher quality. This refers to the educational impact “good” and “bad” teachers have on students. However this quality is determined, there is an argument for equality in terms of an equal distribution of teacher quality across districts (see [7]) and therefore maybe classes. We find this an interesting addition to our design, that could be reflected by

[1] http://www.school-timetabling.com/
[2] http://bach.istc.kobe-u.ac.jp/copris/

an additional property of a teacher. Then we could impose interval constraints on the average or total quality that a class receives. The school executives we talked with told us that there are some teachers that should not teach the same class at the same time. We could not properly implement this without implementing a more complex Boolean interpreter, but the concept of teacher quality could be used as a surrogate.

9 Conclusion

In this thesis we showed how a DSL can abstract from the tedious modelling of constraint satisfaction and optimization problems. Furthermore the constraint language we implemented allows for flexibility in declaring solution goals that. The distinction between hard and weak constraints allows the user to adequatly express her preferences and to consider them in the solving process wherever possible. Our survey shows that the textual representation of a problem as such is generally understood, which is promising for the acceptance by the target audience teachers.
The idea of implementing the different behaviour of individual constraint types in different problem formulations allowed us to formulate the different notions of problems and thereby make use of the strengths of SAT solving and optimization technology. The establishment of a uniform and non-discriminatory efficiency measure in the formulation of the constraint satisfaction problem proved to be a very performant way to get to an efficient solution on which we further apply a local neighborhood search. This is due to the performance advantage SAT solvers have on uniform problems. Application of local search enhances the efficient solution in terms of user preference and establishes an effective trade-off between efficiency and preference.
The resulting solving process is able to quickly find optimal solutions when there are only a few user declared constraints. This allows for an efficient way of exploring the search space by stating possibly conflicting constraints. The generation of feedback for the user establishes a very flexible decision support system rather than only a solving algorithm. The runtime performance decreases with the number of constraints but still seems adequate for a NP-hard problem.
In conclusion, this thesis shows a good example of how a DSL can be designed to hide complex expressions and offer a convenient way to express them anyway.

Bibliography

[1] AMPL Optimization Inc. *AMPL*. Apr. 19, 2016. URL: http://www.ampl.com.

[2] Armin Biere, Marijn Heule, and Hans van Maaren. *Handbook of Satisfiability*. Vol. 185. IOS press, 2009.

[3] Jon A. Breslaw. "A linear programming solution to the faculty assignment problem". In: *Socio-Economic Planning Sciences* 10.6 (1976), pp. 227–230. URL: https://ideas.repec.org/a/eee/soceps/v10y1976i6p227-230.html.

[4] Edmund Kieran Burke and Sanja Petrovic. "Recent research directions in automated timetabling". In: *European Journal of Operational Research* 140.2 (2002), pp. 266–280. URL: http://EconPapers.repec.org/RePEc:eee:ejores:v:140:y:2002:i:2:p:266-280.

[5] Michael W. Carter and Gilbert Laporte. "Practice and Theory of Automated Timetabling II: Second International Conference, PATAT'97 Toronto, Canada, August 20–22, 1997 Selected Papers". In: ed. by Edmund Burke and Michael Carter. Berlin, Heidelberg: Springer Berlin Heidelberg, 1998. Chap. Recent developments in practical course timetabling, pp. 3–19. ISBN: 978-3-540-49803-2. DOI: 10.1007/BFb0055878. URL: http://dx.doi.org/10.1007/BFb0055878.

[6] Raphaël Chenouard, Laurent Granvilliers, and Ricardo Soto. "Model-driven Constraint Programming". In: *Proceedings of the 10th International ACM SIGPLAN Conference on Principles and Practice of Declarative Programming*. PPDP '08. Valencia, Spain: ACM, 2008, pp. 236–246. URL: http://doi.acm.org/10.1145/1389449.1389479.

[7] Lora Cohen-Vogel and La'Tara Osborne-Lampkin. "Allocating Quality: Collective Bargaining Agreements and Administrative Discretion Over Teacher Assignment". In: *Educational Administration Quarterly* 43.4 (2007), pp. 433–461. URL: http://eaq.sagepub.com/content/43/4/433.

[8] Claudia D'Ambrosio and Andrea Lodi. "Mixed integer nonlinear programming tools: a practical overview". In: *4OR* 9.4 (2011), pp. 329–349. URL: http://dx.doi.org/10.1007/s10288-011-0181-9.

[9] Niklas Eén and Niklas Sorensson. "Translating pseudo-boolean constraints into SAT". In: *Journal on Satisfiability, Boolean Modeling and Computation* 2 (2006), pp. 1–26.

[10] Eugene C Freuder and Alan K Mackworth. "Constraint satisfaction: An emerging paradigm". In: *Handbook of Constraint Programming*. Elsevier, 2006, pp. 11–25.

[11] GAMS Development Corporation. *GAMS*. Apr. 19, 2016. URL: http://www.gams.com.

[12] Aldy Gunawan, Kien Ming Ng, and Kim Leng Poh. "Solving the Teacher Assignment-Course Scheduling Problem by a Hybrid Algorithm". In: *International Journal of Mechanical, Aerospace, Industrial, Mechatronic and Manufacturing Engineering* 1.9 (2007), pp. 491–496. URL: http://waset.org/Publications?p=9.

[13] S. Heipcke. "Comparing Constraint Programming and Mathematical Programming Approaches to Discrete Optimisation - The Change Problem". In: *The Journal of the Operational Research Society* 50.6 (1999), pp. 581–595. URL: http://www.jstor.org/stable/3010615.

[14] Holger H. Hoos and Edward P. K. Tsang. "Local Search Methods". In: *Handbook of Constraint Programming*. New York, NY, USA: Elsevier Science Inc., 2006, pp. 245–277. URL: http://dx.doi.org/10.1016/S1574-6526(06)80009-X.

[15] Daniel Le Berre and Anne Parrain. “The SAT4J library, Release 2.2, System Description”. In: *Journal on Satisfiability, Boolean Modeling and Computation* 7 (2010), pp. 59–64. URL: https://hal.archives-ouvertes.fr/hal-00868136.

[16] Pedro Meseguer, Francesca Rossi, and Thomas Schiex. “Soft Constraints”. In: *Handbook of Constraint Programming*. New York, NY, USA: Elsevier Science Inc., 2006, pp. 279–326. URL: http://dx.doi.org/10.1016/S1574-6526(06)80013-1.

[17] MetaBorg Software Foundation. *Spoofax*. Apr. 20, 2016. URL: http://http://www.metaborg.org/spoofax/.

[18] Nicholas Nethercote et al. “MiniZinc: Towards a Standard CP Modelling Language”. In: *Principles and Practice of Constraint Programming – CP 2007: 13th International Conference, CP 2007, Providence, RI, USA, September 23-27, 2007. Proceedings*. Ed. by Christian Bessière. Berlin, Heidelberg: Springer Berlin Heidelberg, 2007, pp. 529–543. URL: http://dx.doi.org/10.1007/978-3-540-74970-7_38.

[19] Optimisation Research Group. *MiniZinc*. Apr. 10, 2016. URL: http://www.minizinc.org/.

[20] Francesca Rossi, Peter van Beek, and Toby Walsh. *Handbook of Constraint Programming (Foundations of Artificial Intelligence)*. New York, NY, USA: Elsevier Science Inc., 2006.

[21] Takehide Soh, Naoyuki Tamura, and Mutsunori Banbara. “Scarab: A Rapid Prototyping Tool for SAT-Based Constraint Programming Systems”. In: *Theory and Applications of Satisfiability Testing – SAT 2013: 16th International Conference, Helsinki, Finland, July 8-12, 2013. Proceedings*. Ed. by Matti Järvisalo and Allen Van Gelder. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, pp. 429–436. URL: http://dx.doi.org/10.1007/978-3-642-39071-5_34.

[22] P.I. Tillett. “An operations research approach to the assignment of teachers to courses”. In: *Socio-Economic Planning Sciences* 9.3 (1975), pp. 101–104. URL: http://dx.doi.org/10.1016/0038-0121(75)90018-X.

[23] Yen-Zen Wang. “An application of genetic algorithm methods for teacher assignment problems”. In: *Expert Systems with Applications* 22.4 (2002), pp. 295–302. URL: http://www.sciencedirect.com/science/article/pii/S0957417402000179.

A Appendix

A.1 Problem Specification

```
problem UV
Subjects
      Subject deutsch
      Subject mathematik
      Subject englisch
      Subject geschichte
      Subject physik
      Subject biologie
      Subject chemie
      Subject franzoesisch
      Subject powi
      Subject sport
      Subject spanisch
      Subject arbeitslehre
      Subject erdkunde
      Subject kunst
      Subject musik
      Subject ethik
      Subject religionkath
      Subject religionev
      Subject powibili
      Subject ekbili
      Subject gbili
      Subject latein
      Subject darstellendes
      Subject informatik
      Lesson without subject qualification klassenstunde
      Lesson LRR with possible subject qualifications: deutsch
      Lesson without subject qualification KO
      Lesson without subject qualification TUT
      Lesson without subject qualification Verbuende
      Lesson without subject qualification REFLEXION
      Lesson without subject qualification PRAKTIKUM
      Lesson without subject qualification LQ
      Lesson without subject qualification sonderdienst
 Lessonplans
  Lessonplan Hauptschule grade 5:
    deutsch -> 5
    englisch -> 5
    kunst -> 2
    arbeitslehre -> 2
    erdkunde -> 2
    ethik -> 2
    mathematik -> 5
    biologie -> 1
    sport -> 2
    klassenstunde -> 1
   Lessonplan Hauptschule grade 6 :
    deutsch -> 5
    englisch -> 5
    kunst -> 2
    arbeitslehre -> 2
    erdkunde -> 2
    geschichte -> 1
    mathematik -> 5
    biologie -> 2
    sport -> 2
    klassenstunde -> 1
    Lessonplan Hauptschule grade 7 :
    deutsch -> 4
    englisch -> 3
    kunst -> 2
```

```
63      arbeitslehre -> 3
64      geschichte -> 2
65      powi -> 2
66      ethik -> 2
67      mathematik -> 4
68      physik -> 2
69      biologie -> 2
70      sport -> 2
71      LRR -> 1
72      klassenstunde -> 0
73     Lessonplan Hauptschule grade 8 :
74      deutsch -> 4
75      englisch -> 3
76      kunst -> 1
77      musik -> 1
78      arbeitslehre -> 5
79      erdkunde -> 2
80      powi -> 2
81      ethik -> 2
82      mathematik -> 4
83      physik -> 2
84      chemie -> 2
85      sport -> 3
86      klassenstunde -> 0
87     Lessonplan Hauptschule grade 9 :
88      deutsch -> 4
89      englisch -> 3
90      kunst -> 2
91      arbeitslehre -> 3
92      erdkunde -> 1
93      geschichte -> 2
94      ethik -> 2
95      mathematik -> 4
96      physik -> 2
97      chemie -> 2
98      biologie -> 2
99      sport -> 2
100     klassenstunde -> 0
101    Lessonplan Realschule grade 5 :
102     deutsch -> 5
103     englisch -> 5
104     mathematik -> 4
105     sport -> 3
106     kunst -> 2
107     biologie -> 2
108     erdkunde -> 2
109     arbeitslehre -> 2
110     klassenstunde -> 1
111    Lessonplan Realschule grade 6 :
112     deutsch -> 5
113     englisch -> 5
114     mathematik -> 4
115     sport -> 2
116     musik -> 2
117     biologie -> 2
118     erdkunde -> 2
119     arbeitslehre -> 2
120     geschichte -> 2
121     LQ -> 1
122    Lessonplan Realschule grade 7 :
123     deutsch -> 4
124     englisch -> 4
125     mathematik -> 4
126     sport -> 2
127     kunst -> 2
128     biologie -> 2
129     physik -> 2
130     powi -> 2
131     klassenstunde -> 0
132    Lessonplan Realschule grade 8 :
133     deutsch -> 3
134     englisch -> 4
```

```
135       mathematik -> 4
136       sport -> 2
137       musik -> 2
138       chemie -> 2
139       physik -> 2
140       arbeitslehre -> 2
141       geschichte -> 2
142       klassenstunde -> 0
143      Lessonplan Realschule grade 9 :
144       deutsch -> 4
145       englisch -> 3
146       mathematik -> 4
147       sport -> 2
148       kunst -> 2
149       chemie -> 2
150       biologie -> 2
151       powi -> 2
152       geschichte -> 2
153       arbeitslehre -> 2
154       sport -> 2
155       klassenstunde -> 0
156      Lessonplan Realschule grade 10 :
157       deutsch -> 4
158       englisch -> 3
159       mathematik -> 4
160       sport -> 2
161       musik -> 2
162       chemie -> 2
163       erdkunde -> 2
164       powi -> 2
165       geschichte -> 2
166       physik -> 2
167       klassenstunde -> 0
168       Lessonplan Gymnasium grade 5 :
169         deutsch -> 5
170       englisch -> 5
171       kunst -> 2
172       musik -> 2
173       erdkunde -> 2
174       mathematik -> 4
175       biologie -> 2
176       sport -> 3
177       klassenstunde -> 1
178       Lessonplan Gymnasium grade 6 :
179         deutsch -> 5
180       englisch -> 5
181       kunst -> 2
182       musik -> 2
183       mathematik -> 4
184       biologie -> 2
185       sport -> 2
186       LQ -> 1
187       geschichte -> 2
188       Lessonplan Gymnasium grade 7 :
189         deutsch -> 5
190       englisch -> 4
191       kunst -> 2
192       mathematik -> 4
193       physik -> 2
194       biologie -> 2
195       sport -> 2
196       klassenstunde -> 0
197       powi -> 2
198       Lessonplan Gymnasium grade 8 :
199        deutsch -> 3
200       englisch -> 4
201       musik -> 2
202       mathematik -> 4
203       physik -> 2
204       chemie -> 2
205       sport -> 2
206       erdkunde -> 1
```

```
207       klassenstunde -> 0
208       geschichte -> 2
209       powi -> 1
210       Lessonplan Gymnasium grade 9 :
211         deutsch -> 4
212       englisch -> 3
213       mathematik -> 4
214       physik -> 1
215       chemie -> 2
216       biologie -> 2
217       sport -> 2
218       erdkunde -> 1
219       klassenstunde -> 0
220       powi -> 2
221       geschichte -> 2
222       Lessonplan Gymnasium grade 10 :
223         deutsch -> 4
224       englisch -> 4
225       biologie -> 2
226       chemie -> 2
227       physik -> 2
228       sport -> 2
229       mathematik -> 5
230       klassenstunde -> 0
231       powi -> 2
232       geschichte -> 2
233   Special Assignments
234         ████  : 2 :  ████
235         ████  : 3 :  ████
236         ████  : 3 :  ████
237         ████  : 1 :  ████
238         ████  : 2 :  ████
239         ████  : 2 :  ████
240         ████  : 1 :  ████
241         ████  : 0,5 :  ███
242         ████  : 1 :  ████
243         ████  : 3 :  ████
244         ████  : 1 :  ████
245         ████  : 3 :  ████
246         ████  : 1 :  ████
247         ████  : 1 :  ████
248         ████  : 3 :  ████
249         ████  : 8 :  ████
250         ████  : 1,5 :  ███
251         ████  : 0,5 :  ███
252         ████  : 2 :  ████
253         ████  : 2 :  ████
254         ████  : 0,5 :  ███
255         ████  : 8 :  ████
256         ████  : 0,5 :  ███
257         ████  : 1 :  ████
258         ████  : 1 :  ████
259         ████  : 10 :  ████
260         ████  : 22 :  ████
261         ████  : 2 :  ████
262         ████  : 6 :  ████
263         ████  : 1 :  ████
264         ████  : 2 :  ████
265         ████  : 6 :  ████
266
267   Hauptschule
268         Class 5a with deviation from lessonplan:
269          KO -> 2 with 2 teachers
270         Class 6a with deviation from lessonplan:
271          KO -> 2 with 2 teachers
272         Class 6b with deviation from lessonplan:
273           KO -> 2 with 3 teachers
274         Common course for classes 6a,6b: ethik -> 2
275         Class 7a with deviation from lessonplan:
276          arbeitslehre -> 2
277          arbeitslehre -> 2
278         Class 8a
```

```
279         Class 8b with deviation from lessonplan:
280          deutsch -> 4
281          mathematik -> 4
282          englisch -> 3
283          powi -> 2
284          kunst -> 2
285          biologie -> 2
286          sport -> 2
287          REFLEXION -> 1
288          ethik -> 2
289          KO -> 2 with 2 teachers
290          PRAKTIKUM -> 5
291         Class 9a
292         Class 9b with deviation from lessonplan:
293          englisch -> 3
294          mathematik -> 4
295          REFLEXION -> 2
296          Verbuende -> 8
297          PRAKTIKUM -> 5
298          deutsch -> 4
299          KO -> 2 with 4 teachers
300 Realschule
301         Class 5a
302         Class 5b
303         Class 5c
304         Common course for classes 5a,5b,5c: ethik -> 2
305                 ethik -> 2
306                 religionev -> 2
307                 religionkath -> 2
308                 LRR -> 1
309         Class 6a with deviation from lessonplan: mathematik -> 5
310         Class 6b
311         Class 6c
312         Common course for classes 6a,6b,6c: ethik -> 2
313                 ethik -> 2
314                 religionev -> 2
315                 religionkath -> 2
316                 deutsch -> 1
317         Class 7a
318         Common course for classes 7a: LRR -> 1
319         Class 7b
320         Class 7c
321         Common course for classes 7b,7c: LRR -> 1
322         Common course for classes 7a,7b,7c: ethik -> 2
323                 ethik -> 2
324                 religionev -> 2
325                 religionkath -> 2
326                 arbeitslehre -> 3
327                 sport -> 3
328                 franzoesisch -> 5
329                 kunst -> 3
330                 kunst -> 3
331         Class 8a with deviation from lessonplan:
332          deutsch -> 3 with 2 teachers
333         Class 8b
334         Class 8c
335         Common course for classes 8a,8b,8c: ethik -> 2
336                 ethik -> 2
337                 religionev -> 2
338                 religionkath -> 2
339                 franzoesisch -> 4
340                 kunst -> 2
341                 mathematik -> 2
342                 arbeitslehre -> 2
343                 LRR -> 1
344                 gbili -> 3
345         Class 9a with deviation from lessonplan: erdkunde -> 2
346         Class 9b with deviation from lessonplan:
347          erdkunde -> 2
348         Class 9c
349         Common course for classes 9a,9b,9c: ethik -> 2
350                 ethik -> 2
```

```
351                religionev -> 2
352                religionkath -> 2
353                kunst -> 2
354                franzoesisch -> 3
355                englisch -> 2
356                ethik -> 2
357                musik -> 2
358                LRR -> 1
359                gbili -> 3
360        Class 10a with deviation from lessonplan:
361         klassenstunde -> 1
362        Class 10b
363        Class 10c with deviation from lessonplan:
364         erdkunde -> 2 with 2 teachers
365        Common course for classes 10a,10b,10c: ethik -> 2
366                  ethik -> 2
367                  religionev -> 2
368                  religionkath -> 2
369                  deutsch -> 2
370                  kunst -> 2
371                  powi -> 2
372                  chemie -> 2
373                  franzoesisch -> 3
374                  gbili -> 3
375  Gymnasium
376        Class 5a with deviation from lessonplan: spanisch -> 1
377        Class 5b with deviation from lessonplan:
378         geschichte -> 0
379         gbili -> 2
380        Class 5c
381        Class 5d
382        Class 5e with deviation from lessonplan: biologie -> 4
383        Class 5f with deviation from lessonplan: biologie -> 4
384        Class 5g
385        Common course for classes 5a,5b,5g: ethik -> 2
386                religionev -> 2
387                religionkath -> 2
388        Common course for classes 5c,5d,5e,5f: religionev -> 2
389                  religionev -> 2
390                  religionkath -> 2
391                  ethik -> 2
392                  ethik -> 2
393        Common course for classes 5a,5b,5c,5d,5e,5f,5g: LRR -> 1
394        Class 6a with deviation from lessonplan:
395         spanisch -> 1
396         englisch -> 5
397         sport -> 2 with 2 teachers
398        Class 6b with deviation from lessonplan:
399         geschichte -> 0
400         gbili -> 2
401        Class 6c
402        Class 6d with deviation from lessonplan:
403         biologie -> 2 with 2 teachers
404        Class 6e with deviation from lessonplan: biologie -> 4
405        Class 6f with deviation from lessonplan: biologie -> 4
406        Class 6g
407        Common course for classes 6a,6b,6c,6d: religionev -> 2
408                  religionev -> 2
409                  ethik -> 2
410                  ethik -> 2
411                  religionkath -> 2
412        Common course for classes 6e,6f,6g: religionev -> 2
413                religionev -> 2
414                ethik -> 2
415                religionkath -> 2
416        Common course for classes 6a,6b,6c,6d,6e: LRR -> 1
417        Common course for classes 6a,6b,6c,6d,6e,6f,6g: LRR -> 1
418        Class 7a with deviation from lessonplan:
419         spanisch -> 5
420         sport -> 2 with 2 teachers
421        Class 7b with deviation from lessonplan:
422         geschichte -> 0
```

```
 powi -> 0
 gbili -> 2
 powibili -> 2
Class 7c with deviation from lessonplan:
 geschichte -> 0
 powi -> 0
 gbili -> 2
 powibili -> 2
Class 7d
Class 7e with deviation from lessonplan: physik -> 4
Class 7f
Class 7g
Common course for classes 7a,7b,7d,7g: religionev -> 2
          religionev -> 2
          religionkath -> 2
          ethik -> 2
Common course for classes 7c,7e,7f: religionev -> 2
        religionev -> 2
        religionkath -> 2
        ethik -> 2
Common course for classes 7a,7b,7c,7d,7e,7f,7g: LRR -> 1
Common course for classes 7b,7c,7d,7e,7f,7g: spanisch -> 5
              spanisch -> 5
              spanisch -> 5
              franzoesisch -> 5
              franzoesisch -> 5
              latein -> 5
Class 8a with deviation from lessonplan: spanisch -> 5
Class 8b with deviation from lessonplan:
 englisch -> 4 with 2 teachers
 chemie -> 2 with 2 teachers
 geschichte -> 0
 gbili -> 2
 powi -> 0
 powibili -> 2
 erdkunde -> 2
Class 8c
Class 8d
Class 8e with deviation from lessonplan: chemie -> 4
Class 8f with deviation from lessonplan: chemie -> 4
Class 8g
Common course for classes 8a,8b,8c: religionev -> 2
        religionev -> 2
        religionkath -> 2
        religionkath -> 2
        ethik -> 2
Common course for classes 8d,8e,8f,8g: religionev -> 2
          religionev -> 2
          religionkath -> 2
          religionkath -> 2
          ethik -> 2
          ethik -> 2
Common course for classes 8b,8f,8g: spanisch -> 4
        spanisch -> 4
        franzoesisch -> 4
        latein -> 4
Common course for classes 8c,8d,8e: spanisch -> 4
        spanisch -> 4
        franzoesisch -> 4
        latein -> 4
Common course for classes 8a,8b,8c,8d,8e,8f,8g: LRR -> 1
Common course for classes 8b,8c,8d,8e,8f,8g: franzoesisch -> 1
Class 9a with deviation from lessonplan:
 kunst -> 2
 spanisch -> 4
Class 9b with deviation from lessonplan:
 powi -> 0
 geschichte -> 0
 powibili -> 2
 gbili -> 2
 kunst -> 2
Class 9c with deviation from lessonplan: musik -> 2
```

```
      Class 9e with deviation from lessonplan:
       musik -> 2
       mathematik -> 4 with 2 teachers
       powi -> 2 with 2 teachers
      Class 9f with deviation from lessonplan: musik -> 2
      Common course for classes 9b,9c,9e,9f: spanisch -> 3
                spanisch -> 3
                franzoesisch -> 3
                franzoesisch -> 3
                latein -> 3
                franzoesisch -> 1
      Common course for classes 9a,9b,9c,9e,9f: religionev -> 2
                  religionev -> 2
                  religionev -> 2
                  religionkath -> 2
                  religionkath -> 2
                  ethik -> 2
                  ethik -> 2
                  sport -> 2
                  chemie -> 2
                  physik -> 2
                  informatik -> 2
                  ethik -> 2
                  latein -> 2
      Class 10a with deviation from lessonplan:
       mathematik -> 5 with 2 teachers
      Class 10b with deviation from lessonplan:
       powi -> 0
       powibili -> 2 with 2 teachers
       geschichte -> 0
       gbili -> 3
      Class 10c
      Class 10e with deviation from lessonplan:
       biologie -> 2 with 2 teachers
      Common course for classes 10a,10b,10c,10e: religionev -> 2
                   religionkath -> 2
                   ethik -> 2
                   ethik -> 2
                   erdkunde -> 2
                   erdkunde -> 2
                   informatik -> 2
                   informatik -> 2
                   spanisch -> 4
                   spanisch -> 3
                   spanisch -> 3
                   franzoesisch -> 3
                   franzoesisch -> 3
                   latein -> 3
                   darstellendes -> 2
                   darstellendes -> 2
                   musik -> 2
                   kunst -> 2
                   kunst -> 2
Upper classes
       Grade 12 :
           Grundkurs: sport -> 2
           Grundkurs: sport -> 2
           Grundkurs: sport -> 2
           Leistungskurs: sport -> 5
           Grundkurs: englisch -> 3
           Grundkurs: englisch -> 3
           Grundkurs: englisch -> 3
           Leistungskurs: englisch -> 5
           Grundkurs: chemie -> 3
           Leistungskurs: chemie -> 5
           Leistungskurs: informatik -> 5
           Grundkurs: informatik -> 3
           Grundkurs: erdkunde -> 2
           Leistungskurs: deutsch -> 5
           Grundkurs: deutsch -> 4
           Grundkurs: deutsch -> 4
           Leistungskurs: mathematik -> 5
```

```
            Grundkurs: mathematik -> 4
            Grundkurs: mathematik -> 4
            Grundkurs: mathematik -> 4
            Leistungskurs: geschichte -> 5
            Leistungskurs: biologie -> 5
            Grundkurs: ethik -> 3
            Grundkurs: ethik -> 3
            Grundkurs: religionev -> 3
            Grundkurs: religionkath -> 3
            Grundkurs: franzoesisch -> 3
            Grundkurs: spanisch -> 3
            Grundkurs: spanisch -> 3
            Grundkurs: spanisch -> 4
            Grundkurs: kunst -> 2
            Grundkurs: musik -> 2
            Grundkurs: darstellendes -> 2
            Grundkurs: powi -> 3
            Grundkurs: powi -> 3
            Grundkurs: powi -> 3
            Grundkurs: geschichte -> 3
            Grundkurs: geschichte -> 3
            Grundkurs: gbili -> 3
            Grundkurs: biologie -> 3
            Grundkurs: biologie -> 3
            Grundkurs: physik -> 3
            Sonstiges: TUT -> 1
            Sonstiges: TUT -> 1
            Sonstiges: TUT -> 1
            Sonstiges: TUT -> 1
            Sonstiges: TUT -> 1
        Grade 13 : Grundkurs: sport -> 2
            Grundkurs: sport -> 2
            Grundkurs: sport -> 2
            Leistungskurs: sport -> 5
            Grundkurs: englisch -> 3
            Grundkurs: englisch -> 3
            Leistungskurs: englisch -> 5
            Leistungskurs: englisch -> 5
            Grundkurs: chemie -> 3
            Leistungskurs: chemie -> 5
            Leistungskurs: informatik -> 5
            Grundkurs: erdkunde -> 2
            Leistungskurs: deutsch -> 5
            Grundkurs: deutsch -> 4
            Grundkurs: deutsch -> 4
            Leistungskurs: mathematik -> 5
            Grundkurs: mathematik -> 4
            Grundkurs: mathematik -> 4
            Grundkurs: mathematik -> 4
            Leistungskurs: geschichte -> 5
            Leistungskurs: biologie -> 5
            Grundkurs: ethik -> 3
            Grundkurs: ethik -> 3
            Grundkurs: religionev -> 3
            Grundkurs: religionkath -> 3
            Grundkurs: franzoesisch -> 3
            Grundkurs: spanisch -> 3
            Leistungskurs: spanisch -> 5
            Grundkurs: kunst -> 2
            Grundkurs: musik -> 2
            Grundkurs: darstellendes -> 2
            Grundkurs: powi -> 3
            Grundkurs: powi -> 3
            Grundkurs: powi -> 3
            Grundkurs: geschichte -> 3
            Grundkurs: geschichte -> 3
            Grundkurs: gbili -> 3
            Grundkurs: biologie -> 3
            Grundkurs: biologie -> 3
            Grundkurs: physik -> 3
            Sonstiges: TUT -> 1
            Sonstiges: TUT -> 1
```

```
639           Sonstiges: TUT -> 1
640           Sonstiges: TUT -> 1
641           Sonstiges: TUT -> 1
```

Listing A.1: Reference Problem Specification

A.2 Realistic Constraints

```
1  Constraints
2    Hard Constraint:
3      Workreduction of teacher [redacted] = 27
4    Weak Constraint with weight 2:
5      For each teacher X in teachers with qualification ekbili
6        and each class Y in classes with subject ekbili:
7          If teacher X teaches class Y in subject ekbili,
8              then teacher X teaches class Y in subject englisch
9    Weak Constraint with weight 2:
10     For each teacher X in teachers with qualification powibili
11       and each class Y in classes with subject powibili:
12         If teacher X teaches class Y in subject powibili,
13             then teacher X teaches class Y in subject englisch
14   Weak Constraint with weight 2:
15     For each teacher X in teachers with qualification gbili
16       and each class Y in classes with subject gbili:
17         If teacher X teaches class Y in subject gbili,
18             then teacher X teaches class Y in subject englisch
19
20   Hard Constraint:
21     Assign: teacher [redacted] teaches class Gymnasium 5a in subject klassenstunde,LQ
22   Hard Constraint:
23     Assign: teacher [redacted] teaches class Gymnasium 5b in subject klassenstunde,LQ
24   Hard Constraint:
25     Assign: teacher [redacted] teaches class Gymnasium 5c in subject klassenstunde,LQ
26   Hard Constraint:
27     Assign: teacher [redacted] teaches class Gymnasium 5d in subject klassenstunde,LQ
28   Hard Constraint:
29     Assign: teacher [redacted] teaches class Gymnasium 5e in subject klassenstunde,LQ
30   Hard Constraint:
31     Assign: teacher [redacted] teaches class Gymnasium 5f in subject klassenstunde,LQ
32   Hard Constraint:
33     Assign: teacher [redacted] teaches class Gymnasium 5g in subject klassenstunde,LQ
34   Hard Constraint:
35     Assign: teacher [redacted] teaches class Gymnasium 6a in subject klassenstunde,LQ
36   Hard Constraint:
37     Assign: teacher [redacted] teaches class Gymnasium 6b in subject klassenstunde,LQ
38   Hard Constraint:
39     Assign: teacher [redacted] teaches class Gymnasium 6c in subject klassenstunde,LQ
40   Hard Constraint:
41     Assign: teacher [redacted] teaches class Gymnasium 6d in subject klassenstunde,LQ
42   Hard Constraint:
43     Assign: teacher [redacted] teaches class Gymnasium 6e in subject klassenstunde,LQ
44   Hard Constraint:
45     Assign: teacher [redacted] teaches class Gymnasium 6f in subject klassenstunde,LQ
46   Hard Constraint:
47     Assign: teacher [redacted] teaches class Gymnasium 6g in subject klassenstunde,LQ
48   Hard Constraint:
49     Assign: teacher [redacted] r teaches class Gymnasium 7a in subject klassenstunde,LQ
50   Hard Constraint:
51     Assign: teacher [redacted] teaches class Gymnasium 7b in subject klassenstunde,LQ
52   Hard Constraint:
53     Assign: teacher [redacted] teaches class Gymnasium 7c in subject klassenstunde,LQ
54   Hard Constraint:
55     Assign: teacher [redacted] teaches class Gymnasium 7d in subject klassenstunde,LQ
56   Hard Constraint:
57     Assign: teacher [redacted] teaches class Gymnasium 7f in subject klassenstunde,LQ
58   Hard Constraint:
59     Assign: teacher [redacted] teaches class Gymnasium 7g in subject klassenstunde,LQ
60   Hard Constraint:
61     Assign: teacher [redacted] teaches class Gymnasium 8a in subject klassenstunde,LQ
62   Hard Constraint:
```

```
    Assign: teacher [redacted] teaches class Gymnasium 8b in subject klassenstunde,LQ
  Hard Constraint:
    Assign: teacher [redacted] teaches class Gymnasium 8c in subject klassenstunde,LQ
  Hard Constraint:
    Assign: teacher [redacted] teaches class Gymnasium 8d in subject klassenstunde,LQ
  Hard Constraint:
    Assign: teacher [redacted] teaches class Gymnasium 8e in subject klassenstunde,LQ
  Hard Constraint:
    Assign: teacher [redacted] teaches class Gymnasium 8f in subject klassenstunde,LQ
  Hard Constraint:
    Assign: teacher [redacted] teaches class Gymnasium 8g in subject klassenstunde,LQ
  Hard Constraint:
    Assign: teacher [redacted] teaches class Gymnasium 9a in subject klassenstunde,LQ
  Hard Constraint:
    Assign: teacher [redacted] teaches class Gymnasium 9b in subject klassenstunde,LQ
  Hard Constraint:
    Assign: teacher [redacted] teaches class Gymnasium 9c in subject klassenstunde,LQ
  Hard Constraint:
    Assign: teacher [redacted] teaches class Gymnasium 9e in subject klassenstunde,LQ
  Hard Constraint:
    Assign: teacher [redacted] teaches class Gymnasium 9f in subject klassenstunde,LQ
  Hard Constraint:
    Assign: teacher [redacted] teaches class Gymnasium 10a in subject klassenstunde,LQ
  Hard Constraint:
    Assign: teacher [redacted] teaches class Gymnasium 10b in subject klassenstunde,LQ
  Hard Constraint:
    Assign: teacher [redacted] teaches class Gymnasium 10c in subject klassenstunde,LQ
  Hard Constraint:
    Assign: teacher [redacted] teaches class Gymnasium 10e in subject klassenstunde,LQ
  Hard Constraint:
    Assign: teacher [redacted] teaches class Realschule 5a in subject klassenstunde,LQ

  Weak Constraint with weight 2:
    For each class Y in classes with subject klassenstunde :
        For each teacher X in all teachers :
            If teacher X teaches class Y in subject klassenstunde,
                then teacher X teaches class Y in all subjects except LRR
  Weak Constraint with weight 2:
    For each class Y in classes with subject LQ :
        For each teacher X in all teachers :
            If teacher X teaches class Y in subject LQ,
                then teacher X teaches class Y in all subjects except LRR
  Weak Constraint with weight 2:
    For each class Y in classes with subject LRR :
        For each teacher X in teachers with qualification deutsch:
            If teacher X teaches class Y in subject deutsch,
                then teacher X teaches class Y in subject LRR

Deputations
  Deputation Oberstufe for teacher X when Workload of assignments where teacher X teaches classes
    in grade 12,13 >= 7
```

Listing A.2: Realistic Constraints and Deputations

TeachAlloc Evaluierung

Wir haben eine Programmiersprache entwickelt, mit der die Personalplanung an Schulen modelliert werden kann. In dieser Befragung zeigen wir Ihnen kleine Beispielplanungen oder Ausschnitte einer Planung und fragen Ihr Verständnis der Formulierungen ab.
Wir danken Ihnen dafür, dass Sie sich die Zeit nehmen, die folgenden Fragen zu beantworten. Dies sollte maximal 15 Minuten in Anspruch nehmen. Ihre Eingaben sind selbstverständlich anonym.

* Erforderlich

Stundentafel

Eine Stundentafel gibt an, in welchen Fächern und in jeweils wie vielen Stunden eine Klasse einer bestimmten Klassenstufe unterrichtet werden soll.

```
Stundentafel Hauptschule Klassenstufe 5:
   deutsch -> 5
   englisch -> 5
   mathematik -> 5
   geschichte -> 2
   klassenlehrerstunde -> 1
```

1. **Wie viele Unterrichtsstunden Englisch hat eine 5. Klasse im Hauptschulzweig?** *

   .....

Klassen und Kurse

In jedem Schulzweig können Klassen definiert werden. Normalerweise wird ihnen die jeweilige Stundentafel zugewiesen.

```
Stundentafeln
   Stundentafel Hauptschule Klassenstufe 5:
      deutsch -> 5
      englisch -> 5
      mathematik -> 5
      geschichte -> 2
      klassenlehrerstunde -> 1

Hauptschule
   Klasse 5a
```

2. **Wie viele Unterrichtsstunden hat Klasse 5a im Fach Geschichte?** *

   Markieren Sie nur ein Oval.

   - ◯ 0
   - ◯ 2
   - ◯ 4
   - ◯ 6

Klassen und Kurse

Manche Klassen weichen von der Stundentafel aber ab.

```
Stundentafeln
   Stundentafel Hauptschule Klassenstufe 5:
      deutsch -> 5
      englisch -> 5
      mathematik -> 5
      geschichte -> 2
      klassenlehrerstunde -> 1

Hauptschule
   Klasse 5b mit Abweichung von Stundentafel:
      geschichte -> 4
```

3. **Wie viele Unterrichtsstunden hat Klasse 5b im Fach Geschichte?** *

   Markieren Sie nur ein Oval.

   - ◯ 0
   - ◯ 2
   - ◯ 4
   - ◯ 6

Klassen und Kurse

Einzelne Kurse werden nicht nur für Schüler einer bestimmten Klasse gegeben, sondern für Schüler mehrerer Klassen.

```
Stundentafeln
   Stundentafel Hauptschule Klassenstufe 5:
      deutsch -> 5
      englisch -> 5
      mathematik -> 5
      geschichte -> 2
      klassenlehrerstunde -> 1

Hauptschule
   Gemeinsame Kurse in Klassen 5a,5b,5c:
      ethik -> 2
      franzoesisch -> 2
```

4. **Wie viele Unterrichtsstunden gibt es insgesamt im Fach Ethik?** *

   Markieren Sie nur ein Oval.

   - ◯ 0
   - ◯ 2
   - ◯ 4
   - ◯ 6

Klassen und Kurse

In jedem Schulzweig können Klassen definiert werden. Normalerweise wird ihnen die jeweilige Stundentafel zugewiesen. Manche Klassen weichen von der Stundentafel aber ab.
Einzelne Kurse werden nicht nur für Schüler einer bestimmten Klasse gegeben, sondern für Schüler mehrerer Klassen.

```
Stundentafeln
   Stundentafel Hauptschule Klassenstufe 5:
      deutsch -> 5
      englisch -> 5
      mathematik -> 5
      geschichte -> 2
      klassenlehrerstunde -> 1

Hauptschule
   Klasse 5a

   Klasse 5b mit Abweichung von Stundentafel:
      geschichte -> 4

   Klasse 5c mit Abweichung von Stundentafel:
      geschichte -> 0
      geschichte_bilingual -> 2

   Kurse in Klassen 5a+5b+5c:
      ethik -> 2
      franzoesisch -> 2
```

5. **Wie viele Unterrichtsstunden gibt es insgesamt im Fach Deutsch?** *
   Markieren Sie nur ein Oval.

   - ◯ 10
   - ◯ 15
   - ◯ 20

Lehrer

Lehrer sind entweder dem Gymnasial- oder Haupt-/Realschuldienst zugeordnet. Ihre Qualifikationen besagen, welche Fächer sie in welchen Schulzweigen unterrichten dürfen. Gymnasiallehrer dürfen ihre Fächer auch grundsätzlich im Zweig Haupt-/Realschule lehren. Die zu unterrichtenden Stunden sind gesetzlich und vertraglich individuell festgelegt.

```
Lehrer
   Lehrer Ackermann
         Dienst : GYM
         Fachqualifikation :
            Gymnasium: deutsch,ethik
         Pflichtstunden : 26

   Lehrer Dittrich
         Dienst : HR
         Fachqualifikation :
            Haupt-/Realschule: geschichte,englisch,geschichte_bilingual
         Pflichtstunden : 20

   Lehrer Genscher
         Dienst : HR
         Fachqualifikation :
            Haupt-/Realschule: mathematik,englisch
         Pflichtstunden : 25

   Lehrer Tauber
         Dienst : GYM
         Fachqualifikation :
            Gymnasium: franzoesisch,deutsch
         Pflichtstunden : 26
```

6. **Welche Fächer unterrichtet Lehrer Genscher?** *
   Wählen Sie alle zutreffenden Antworten aus.

   - [ ] Deutsch
   - [ ] Englisch
   - [ ] Geschichte
   - [ ] Mathematik

7. **Welche Fächer können im Gymnasialzweig überhaupt unterrichtet werden?** *
   Wählen Sie alle zutreffenden Antworten aus.

   - [ ] Deutsch
   - [ ] Englisch
   - [ ] Mathematik
   - [ ] Französisch
   - [ ] Ethik

Lehrer und Klassen

Lehrer sind entweder dem Gymnasial- oder Haupt-/Realschuldienst zugeordnet. Ihre Qualifikationen besagen, welche Fächer sie in welchen Schulzweigen unterrichten dürfen. Gymnasiallehrer dürfen ihre Fächer auch grundsätzlich im Zweig Haupt-/Realschule lehren. Die zu unterrichtenden Stunden sind gesetzlich und vertraglich individuell festgelegt.

```
Hauptschule
   Klasse 5a

   Klasse 5b mit Abweichung von Stundentafel:
      geschichte -> 4

Gymnasium
   Klasse 7a

   Klasse 7b

Lehrer
   Lehrer Ackermann
         Dienst : GYM
         Fachqualifikation :
            Gymnasium: deutsch,ethik
         Pflichtstunden : 26
```

8. **Kann Lehrer Ackermann Klasse 5a in Deutsch unterrichten?** *
   Markieren Sie nur ein Oval.

   - ◯ ja
   - ◯ nein

Lehrer und Klassen

Lehrer sind entweder dem Gymnasial- oder Haupt-/Realschuldienst zugeordnet. Ihre Qualifikationen besagen, welche Fächer sie in welchen Schulzweigen unterrichten dürfen. Gymnasiallehrer dürfen ihre Fächer auch grundsätzlich im Zweig Haupt-/Realschule lehren. Die zu unterrichtenden Stunden sind gesetzlich und vertraglich individuell festgelegt.

```
Hauptschule
   Klasse 5a

   Klasse 5b mit Abweichung von Stundentafel:
      geschichte -> 4

Gymnasium
   Klasse 7a

   Klasse 7b

Lehrer
   Lehrer Dittrich
         Dienst : HR
         Fachqualifikation :
            Haupt-/Realschule: geschichte,englisch,geschichte_bilingual
         Pflichtstunden : 20
```

9. **Kann Lehrer Dittrich Klasse 7b in Geschichte unterrichten?** *

   Markieren Sie nur ein Oval.

   - ja
   - nein

Lehrer und Klassen

Lehrer sind entweder dem Gymnasial- oder Haupt-/Realschuldienst zugeordnet. Ihre Qualifikationen besagen, welche Fächer sie in welchen Schulzweigen unterrichten dürfen. Gymnasiallehrer dürfen ihre Fächer auch grundsätzlich im Zweig Haupt-/Realschule lehren. Die zu unterrichtenden Stunden sind gesetzlich und vertraglich individuell festgelegt.

```
Hauptschule
   Klasse 5a

   Klasse 5b mit Abweichung von Stundentafel:
      geschichte -> 4

Gymnasium
   Klasse 7a

   Klasse 7b

Lehrer
   Lehrer Genscher
         Dienst : HR
         Fachqualifikation :
            Haupt-/Realschule: mathematik,englisch
         Pflichtstunden : 25
```

10. **Kann Lehrer Genscher Klasse 5b in Deutsch unterrichten?** *

    Markieren Sie nur ein Oval.

    - ja
    - nein

Lehrer und Klassen

Lehrer sind entweder dem Gymnasial- oder Haupt-/Realschuldienst zugeordnet. Ihre Qualifikationen besagen, welche Fächer sie in welchen Schulzweigen unterrichten dürfen. Gymnasiallehrer dürfen ihre Fächer auch grundsätzlich im Zweig Haupt-/Realschule lehren. Die zu unterrichtenden Stunden sind gesetzlich und vertraglich individuell festgelegt.

```
Hauptschule
   Klasse 5a

   Klasse 5b mit Abweichung von Stundentafel:
      geschichte -> 4

Gymnasium
   Klasse 7a

   Klasse 7b

Lehrer
   Lehrer Tauber
         Dienst : GYM
         Fachqualifikation :
            Gymnasium: franzoesisch,deutsch
         Pflichtstunden : 26
```

11. **Kann Lehrer Tauber Klasse 5a in Deutsch unterrichten?** *

Markieren Sie nur ein Oval.

- ja
- nein

Vorgaben

Damit nicht nur irgendeine Zuweisung von Lehrern zu Klassen gefunden wird, sondern eine möglichst gute, können auch Vorgaben für den Lösungsprozess angegeben werden.
Harte Vorgaben müssen befolgt werden. Ist das nicht möglich, wird ein Fehler gemeldet.
Ein Gütekriterium ist die Abweichung von den Pflichtstunden eines Lehrers.

```
Lehrer
   Lehrer Ackermann
         Dienst : GYM
         Fachqualifikation :
            Gymnasium: deutsch,ethik
         Pflichtstunden : 26

Vorgaben
   Harte Vorgabe:
      Überstunden von Lehrer Ackermann <= 1
      Stundenreduzierung von Lehrer Ackermann <= 3
```

12. **Wie viele Stunden muss Lehrer Ackermann mindestens unterrichten?** *

Markieren Sie nur ein Oval.

- 0
- 23
- 24
- 25
- 26

13. **Wie viele Stunden muss Lehrer Ackermann höchstens unterrichten?** *

Markieren Sie nur ein Oval.

- ◯ 26
- ◯ 27
- ◯ 28
- ◯ 29

Vorgaben

Damit nicht nur irgendeine Zuweisung von Lehrern zu Klassen gefunden wird, sondern eine möglichst gute, können auch Vorgaben für den Lösungsprozess angegeben werden.
Harte Vorgaben müssen befolgt werden. Ist das nicht möglich, wird ein Fehler gemeldet.
Ein Gütekriterium ist die Abweichung von den Pflichtstunden eines Lehrers.

```
Lehrer
   Lehrer Tauber
         Dienst : GYM
         Fachqualifikation :
            Gymnasium: franzoesisch,deutsch
         Pflichtstunden : 26

Vorgaben
   Harte Vorgabe:
      Überstunden von Lehrer Tauber = 0
```

14. **Wie viele Stunden muss Lehrer Tauber mindestens unterrichten?** *

Markieren Sie nur ein Oval.

- ◯ 0
- ◯ 25
- ◯ 26

Vorgaben

Damit nicht nur irgendeine Zuweisung von Lehrern zu Klassen gefunden wird, sondern eine möglichst gute, können auch Vorgaben für den Lösungsprozess angegeben werden.
Harte Vorgaben müssen befolgt werden. Ist das nicht möglich, wird ein Fehler gemeldet.

```
Lehrer
   Lehrer Ackermann
         Dienst : GYM
         Fachqualifikation :
            Gymnasium: deutsch,ethik
         Pflichtstunden : 26

Vorgaben
   Harte Vorgabe:
      Verbiete: Lehrer Ackermann unterrichtet Unterricht ethik
```

15. **Kann Lehrer Ackermann eine Klasse in Ethik unterrichten?** *

Markieren Sie nur ein Oval.

- ◯ ja
- ◯ nein

Vorgaben

Damit nicht nur irgendeine Zuweisung von Lehrern zu Klassen gefunden wird, sondern eine möglichst gute, können auch Vorgaben für den Lösungsprozess angegeben werden.
Harte Vorgaben müssen befolgt werden. Ist das nicht möglich, wird ein Fehler gemeldet.

```
Lehrer

   Lehrer Dittrich
         Dienst : HR
         Fachqualifikation :
            Haupt-/Realschule: geschichte,englisch,geschichte_bilingual
         Pflichtstunden : 20

Vorgaben
 Harte Vorgabe:
  Anzahl an Zuweisungen in denen Lehrer Dittrich Unterricht geschichte unterrichtet >= 2
```

16. **Wie viele Klassen unterrichtet Lehrer Dittrich mindestens im Fach Geschichte?**
    *

Regeln

Vorgaben können nicht nur für bestimmte Lehrer oder Klassen gelten, sondern z.B. für alle Lehrer mit einer bestimmten Eigenschaft.

```
Lehrer
   Lehrer Ackermann
         Dienst : GYM
         Fachqualifikation :
            Gymnasium: deutsch,ethik
         Pflichtstunden : 26

   Lehrer Dittrich
         Dienst : HR
         Fachqualifikation :
            Haupt-/Realschule: geschichte,englisch,geschichte_bilingual
         Pflichtstunden : 20

   Lehrer Genscher
         Dienst : HR
         Fachqualifikation :
            Haupt-/Realschule: mathematik,englisch
         Pflichtstunden : 25

   Lehrer Tauber
         Dienst : GYM
         Fachqualifikation :
            Gymnasium: franzoesisch,deutsch
         Pflichtstunden : 26

Vorgaben
   Harte Vorgabe:
            Für jeden Lehrer X aus Alle Lehrer gilt:
                        Überstunden von Lehrer X <= 2
                        Stundenreduzierung von Lehrer X <= 4

   Harte Vorgabe:
            Für jeden Lehrer X aus Lehrer mit Qualifikation englisch gilt:
                        Überstunden von Lehrer X <= 1
                        Stundenreduzierung von Lehrer X <= 2
```

17. **Für welche Lehrer gilt die harte Vorgabe aus Zeile 27?** *
Wählen Sie alle zutreffenden Antworten aus.

- [ ] Ackermann
- [ ] Dittrich
- [ ] Genscher
- [ ] Tauber

18. **Für welche Lehrer gilt die harte Vorgabe aus Zeile 32?** *
Wählen Sie alle zutreffenden Antworten aus.

- [ ] Ackermann
- [ ] Dittrich
- [ ] Genscher
- [ ] Tauber

Weiche Vorgaben

Weiche Vorgaben werden wenn möglich befolgt. Bei einem Widerspruch zwischen zwei weichen Vorgaben hat die mit höherer Gewichtung Vorrang.

```
Hauptschule
   Klasse 5c mit Abweichung von Stundentafel:
      geschichte -> 0
      geschichte_bilingual -> 2

Lehrer
   Lehrer Ackermann
         Dienst : GYM
         Fachqualifikation :
            Gymnasium: deutsch,ethik
         Pflichtstunden : 26

   Lehrer Dittrich
         Dienst : HR
         Fachqualifikation :
            Haupt-/Realschule: geschichte,englisch,geschichte_bilingual
         Pflichtstunden : 20

   Lehrer Genscher
         Dienst : HR
         Fachqualifikation :
            Haupt-/Realschule: mathematik,englisch
         Pflichtstunden : 25

   Lehrer Tauber
         Dienst : GYM
         Fachqualifikation :
            Gymnasium: franzoesisch,deutsch
         Pflichtstunden : 26

Vorgaben
   Weiche Vorgabe mit Gewichtung 5:
    Setze: Lehrer Genscher unterrichtet Klasse Hauptschule 5c in Unterricht englisch

   Weiche Vorgabe mit Gewichtung 10:
        Für jeden Lehrer X aus Lehrer mit Qualifikation geschichte_bilingual gilt:
            Wenn Lehrer X Klasse 5c in Unterricht geschichte_bilingual unterrichtet,
               dann unterrichtet Lehrer X Klasse 5c in Unterricht englisch
```

19. **Welche Vorgabe hat bei einem Widerspruch Vorrang?** *
Markieren Sie nur ein Oval.

- ◯ Vorgabe in Zeile 32
- ◯ Vorgabe in Zeile 35

20. **Welcher Lehrer unterrichtet Klasse 5c in Englisch?** *
Markieren Sie nur ein Oval.

- ◯ Ackermann
- ◯ Dittrich
- ◯ Genscher
- ◯ Tauber

Mehrere Vorgaben

Eine Lösung enthält die Zuweisungen der Lehrer zu Unterrichtseinheiten. Unten sehen sie die addierten Unterrichtsstunden für einzelne Fächer aus den Zuweisungen einer Lösung für vier Lehrer.

```
Lehrer
   Lehrer Ackermann
         Dienst : GYM
         Fachqualifikation :
            Gymnasium: deutsch,ethik
         Pflichtstunden : 26

   Lehrer Dittrich
         Dienst : HR
         Fachqualifikation :
            Haupt-/Realschule: geschichte,englisch,geschichte_bilingual
         Pflichtstunden : 20

   Lehrer Genscher
         Dienst : HR
         Fachqualifikation :
            Haupt-/Realschule: mathematik,englisch
         Pflichtstunden : 25

   Lehrer Tauber
         Dienst : GYM
         Fachqualifikation :
            Gymnasium: franzoesisch,deutsch
         Pflichtstunden : 26

Vorgaben
   Weiche Vorgabe mit Gewichtung 10:
      Für jeden Lehrer X aus Alle Lehrer gilt:
         Überstunden von Lehrer X <= 1

   Harte Vorgabe:
      Unterrichtsstunden in denen Lehrer Dittrich Unterricht englisch unterrichtet >= 10

   Harte Vorgabe:
      Stundenreduzierung von Lehrer Tauber >= 2

   Harte Vorgabe:
      Verbiete: Lehrer Ackermann unterrichtet Unterricht ethik
```

Lösung

Lehrer\Fach	englisch	deutsch	mathematik	ethik	franzoesisch	geschichte	Summe
Ackermann		24		2			26
Dittrich	21					4	25
Genscher	10		11				21
Tauber		12			12		24

21. **Kreuzen Sie die eingehaltenen Vorgaben an.** *

Wählen Sie alle zutreffenden Antworten aus.

- [ ] Vorgabe in Zeile 27
- [ ] Vorgabe in Zeile 31
- [ ] Vorgabe in Zeile 34
- [ ] Vorgabe in Zeile 37

Abschließende Fragen

22. **Wie alt sind Sie?** *

.....

23. **Haben Sie jemals eine Programmiersprache selbst benutzt? Es zählt jedwede Erfahrung.** *

Markieren Sie nur ein Oval.

- ( ) ja
- ( ) nein

24. **Wenn ja, mit welchen Programmiersprachen haben Sie bereits Erfahrungen gesammelt?**

.....

.....

.....

.....

.....

25. **Sind Sie schon einmal mit den Planungen zur Unterrichtsverteilung beschäftigt gewesen?** *

Markieren Sie nur ein Oval.

- ( ) ja
- ( ) nein