Constraint Logic Programming Approach to Teacher Hour Allocation for University Subjects

Daniel Pereira da Silva and Miguel António Ramalho

Faculdade de Engenharia da Universidade do Porto, Portugal,
Logic Programming - 3MIEIC1 Docente_3
daniel.s@fe.up.pt
m.ramalho@fe.up.pt

Abstract. The aims of this paper are threefold. Firstly, to present a constraint satisfaction problem model for the problem concerning *Teacher Hour Allocation for University Subjects*, in both semesters using the *Prolog* programming language, delving into optimal solution search algorithms. Secondly, it proposes an iterative algorithm that can be used to produce increasingly more complex problems for the solver to handle, such approach is considered to be adaptable to any number of similar problems. Thirdly, we reflect upon a number of issues the presented work has arisen and tries to spur interest in improving the present solver, with applications to other similar constraint logic programming situations.

Keywords: logic programming, Prolog, constraint logic programming, clpfd, SICStus, time allocation

1 Introduction

1.1 Problem Description

Consider a Faculty's department that is faced, on a yearly basis, with the distribution of the hours to be taught, from all the subjects available, to the available teachers. For any given year, each subject is characterized by $\boldsymbol{HT}$ and $\boldsymbol{HP}$ the theoretical and practical hours per week, respectively (typically HT <HP). Each subject belongs to a given field of studies, and so does each teacher. The theoretical hours can only be taught by teachers from the same field, whereas practical hours need not be, but preferably are, taught by teachers from the field.

Each teacher can also be labeled according to their affiliation, that reflects in their average weekly hours for both semesters, which can be 7h, 8h or 9h. Additionally, each teacher can specify their preference of having more hours in one semester rather than the other, in the form of a value $\boldsymbol{Diff}$ that reflects the difference in hours between the first and the second semester (positive values mean the teachers prefers more classes on the first semester, and negative on the second).

1.2 Terminology Reference

In order to better pass along the ideas in this paper we devised a fixed terminology for the problem-specific concepts, as follows:

- Let $\boldsymbol{S(Semester, Field, HT, HP, DT, DP)}$ be a subject, characterized by belonging to a semester $\boldsymbol{Semester}$, a field $\boldsymbol{Field}$, by having $\boldsymbol{HT}$ theoretical hours divided into blocks of $\boldsymbol{DT}$ hours and $\boldsymbol{HP}$ practical hours divided into blocks of $\boldsymbol{DP}$ hours;
- Let $\boldsymbol{T(Avg, Diff, Field, HS1, HS2)}$ be a teacher, characterized by the average $\boldsymbol{Avg}$ number of week hours that should be taught per semester, by reflecting the teacher's preference $\boldsymbol{Diff}$, by having a particular field of expertise $\boldsymbol{Field}$, and by teaching $\boldsymbol{HS1}$ hours in the first semester and $\boldsymbol{HS2}$ in the second semester;
- Let $\boldsymbol{Var.Prop}$ refer to the value of the property $\boldsymbol{Prop}$ of the variable $\boldsymbol{Var}$;
- Let _, when used as a replacement of a parameter in the above tuples to be considered as an irrelevant value.

Additionally, upper case letters are used at the beginning of variables that are not set, and lower case for specific values those variables can assume (honoring the Prolog semantics).

2 Literature Review

The problem at hand can be thought of as a subproblem that many timetabling problems have to face. Although simpler, this set of problems still fits the nondeterministic polynomial time class and cannot, therefore, be tackled with a simple approach. It is not a timetabling problem in the sense that there is no time-line order required for the events, the lessons. Both timetabling problems and its sub-problems, as is the present one, fit in the constraint satisfaction problems - CSPs - and different approaches have been applied to them, from graph theoretical models [6], to greedy and genetic algorithms [7] even in recent times [1], and constraint satisfaction programming techniques [12]. It is on this last approach that we focus, given the problem's nature, and the techniques used are standard for constraint modeling [2].

The ability to express problems of high complexity as a CSP is a step towards a successful constraint logic programming approach, and many problems can indeed be expressed in terms of constraint satisfaction clauses over finite domains [9], in fact successful attempts have been made at simplifying this process using *SICStus Prolog* [3] , the same tool used for this project, and we believe there is still a lot of unexplored raw potential in this area.

As such, we devised a number of constraints that we believe powerful enough, devised an heuristic that reflects the optimization goals and worked through search techniques and present our own approach - in Section 4- that we believe to bring novel and elegant constraints that could, in the future, be used across a larger range of problems. In order to properly test and validate our results, we also devised a generation tool for test instances - an essential part for the validation of the results produced by solvers [7] [8], this is specified in Section 3.

3 Test Case Generation

Although the problem at hand is a parallelism of real world problems, it has some minutenesses that make the task of adapting real world situations as problem instances for our solver, by hand, quite a slow process. One of this minutenesses is the consideration of **Diff** - the difference that expresses a teacher's preference of semester hours bias - as it is rarely observable in real world scenarios, although present in many. So, we opted for an iterative test case generation of consecutively larger solvable problems.

3.1 Iterative Test Case Algorithm

In order to generate a complex valid problem, an abstraction effort was made that resulted in devising an iterative method. Consider the following steps:

1. Assume the existence of the following variables:
   - $\boldsymbol{MaxDiff}$ which describes the maximum value the $\boldsymbol{Diff}$ (Section 1.2) can take for any randomly generated teacher;
   - $\boldsymbol{NFields}$ that describes the maximum value the $\boldsymbol{Field}$ of a teacher, subject can take;
   - $\boldsymbol{TSets}$ a predefined set of pairs $\boldsymbol{(HT, DT)}$ that match possible values for theoretical hours ($\boldsymbol{HT}$) in a subject with corresponding valid durations for them ($\boldsymbol{DT}$);
   - $\boldsymbol{PSets}$ a predefined set of pairs $\boldsymbol{(HP, DP)}$ that match possible values for practical hours ($\boldsymbol{HP}$) in a subject with corresponding valid durations for them ($\boldsymbol{DP}$);
   - $\boldsymbol{Avgs}$ is a list of the usual average hours for teachers, typically 7h, 8h and 9h;
   - $\boldsymbol{SEM}$ represents the current semester and it's value is updated in each iteration;
   - $\boldsymbol{CSEM}$ is always the complementary semester of $\boldsymbol{SEM}$.
2. Generate a subject $\boldsymbol{S1}$:

$$\boldsymbol{S1 = S(SEM, f1, HT, HP, DT, DP)}$$

$$\boldsymbol{0 \leq f1 \leq NFields, (HT, DT) \in TSets, (HP, DP) \in PSets}$$

3. Generate a teacher $\boldsymbol{T1}$ (with the same field as $\boldsymbol{S1}$, $\boldsymbol{Diff}$ is calculated after step 5):

$$\boldsymbol{T1 = T(Avg, Diff, f1, HS1, HS2)}$$

$$\boldsymbol{Avg \in Avgs, Diff = 0, HS1 = 0, HS2 = 0}$$

4. Assign $\boldsymbol{HT}$ of $\boldsymbol{S1}$ to $\boldsymbol{T1}$ (this is always possible due to the aforementioned predefined sets):

$$\boldsymbol{T1 = S(SEM, f1, 0, HP, \_, DP)}$$

$$\boldsymbol{SEM = 1 \Rightarrow T1.HS1 \coloneqq T1.HS1 + S1.HT}$$

$$\boldsymbol{SEM = 2 \Rightarrow T1.HS2 \coloneqq T1.HS2 + S1.HT}$$

5. Calculate $\boldsymbol{AT}$ (AvailableTime the available time for $\boldsymbol{T1}$ in semester $\boldsymbol{SEM}$) from $\boldsymbol{AH}$ (AvailableHours - the amount of hours $\boldsymbol{T1}$ can still teach):

$$\boldsymbol{AH = 2 * T1.Avg - T1.HS1 - T1.HS2}$$

   Consider $\boldsymbol{AT}$ new hours for $\boldsymbol{SEM}$, which means $\boldsymbol{AH - AT}$ still available for $\boldsymbol{CSEM}$. For the first case ($\boldsymbol{SEM = 1}$):

$$\boldsymbol{T1.HS1' - T1.HS2' \leq MaxDiff}$$

$$\boldsymbol{(T1.HS1 + AT) - (T1.HS2 + AH - AT) \leq MaxDiff \Leftrightarrow}$$

$$\boldsymbol{T1.HS1 + AT - T1.HS2 - AH + AT \leq MaxDiff \Leftrightarrow}$$

$$\boldsymbol{AT \leq \frac{MaxDiff - T1.HS1 + T1.HS2 + AH}{2}}$$

For the second case ($\boldsymbol{SEM = 2}$), the result would be:

$$\boldsymbol{AT \leq \frac{MaxDiff - T1.HS2 + T1.HS1 + AH}{2}}$$

This formula yields the maximum amount of time $\boldsymbol{T1}$ could teach in semester $\boldsymbol{SEM}$. Knowing $\boldsymbol{S1.DP}$ means we implicitly know the number of blocks $\boldsymbol{T1}$ can teach in practical hours (as no theoretical hours are left to assign), $\boldsymbol{NB}$ (NumberBlocks):

$$\boldsymbol{NB = ATdivDP}$$

Which allows us to assign $\boldsymbol{NB * S1.DP}$ practical hours to $\boldsymbol{T1}$, according to the current semester, $\boldsymbol{SEM}$:

$$\boldsymbol{S1.HP \coloneqq S1.HP - NB * DP}$$

$$\boldsymbol{SEM = 1 \Rightarrow T1.HS1 \coloneqq T1.HS1 - NB * S1.DP}$$

$$\boldsymbol{SEM = 2 \Rightarrow T1.HS2 \coloneqq T1.HS2 - NB * S1.DP}$$

6. After this assignment, we generate as many teachers, and assign them to $\boldsymbol{S1}$'s practical hours (5), as needed until $\boldsymbol{S1.HP = 0}$. We denote that, due to the nature of the problem, the field of the new teachers needs not be the same as T1. The only requirement is: $\boldsymbol{Avg \in Avgs}$.
   The generated teachers for this iteration (including $\boldsymbol{T1}$) are $\boldsymbol{GenTeachers}$.
   The fields of the teachers in $\boldsymbol{GenTeachers}$ are $\boldsymbol{FGenTeachers}$.
7. As a consequence of the previous steps, there exists what can be named as a $\boldsymbol{Debt}$ of Hours from the teachers that need to be assigned to them in $\boldsymbol{CSEM}$ so that the $\boldsymbol{MaxDiff}$ restriction is respected. The value of the total $\boldsymbol{Debt}$ is calculated as the sum of the individual teacher debts ($\boldsymbol{Debt_i}$ is the debt of teacher $\boldsymbol{T_i \in GenTeachers}$):

$$\boldsymbol{Debt = \sum_{i=0}^{\|GenTeachers\|} |T_i.HS1 - T_i.HS2| - MaxDiff}$$

8. Zero $\boldsymbol{Debt}$ by creating a new subject, $\boldsymbol{S_{Debt}}$:

$$\boldsymbol{S_{Debt} = S(CSEM, Field, 0, Debt, , 1)}$$

$$\boldsymbol{Field \in FGenTeachers}$$

This concludes an iteration of a test case generation. The steps 1 to 8 can be repeated as many times as necessary, and the resulting subjects ($\boldsymbol{S1}$ and $\boldsymbol{S_{Debt}}$) and teachers ($\boldsymbol{GenTeachers}$) can be accumulated into increasingly more complex test cases.

A proof of concept for the above iterative algorithm has been implemented in Python3 and used for testing the solver presented in this paper.

4 Constraint Logic Programming Approach

Using *SICStus Prolog* and, more specifically, its constraint programming module, *clp(FD)*, we devised a solver for the aforementioned problem.

4.1 Decision Variables

In order to apply constraints in an effective and readable manner, we constructed two entity-oriented model containers for the decision variables - one for the teachers ($\boldsymbol{Teachers}$) and another for the subjects ($\boldsymbol{Subjects}$). Denoting that all decision variables are finite domain variables, that is if $\boldsymbol{A}$ is a decision variable $\boldsymbol{A} \in \mathbb{Z}$ holds true:

$\boldsymbol{Teachers}$

This Prolog variable is a list. Each element of this list has the following structure:

```
Teacher = Avg-Diff-Field-HS1-HS2
```

Prolog snippet 1.1: Modeling of the Teacher entity in Prolog

Where $\boldsymbol{Avg}$, $\boldsymbol{Diff}$, $\boldsymbol{Field}$ are initially known values as previously described and $\boldsymbol{HS1}$ and $\boldsymbol{HS2}$ are the decision variables for a given teacher, representing the number of hours taught by the teacher in the first and second semesters, respectively.
The domains for the variables are as follows:

$$\boldsymbol{0 \leq HS1 \leq \lceil \frac{2 * Avg + Diff}{2} \rceil}$$

$$\boldsymbol{0 \leq HS2 \leq \lceil \frac{2 * Avg - Diff}{2} \rceil}$$

As this considers that a teacher can give a minimum of 0 hours and a maximum that depends on the preference manifested through $\boldsymbol{Diff}$. Using this domain instead of the more immediate one - $\boldsymbol{2 * Avg}$ - yields results more efficiently as it immediately excludes redundant values from the domain.

$\boldsymbol{Subjects}$

This Prolog variable is a list. Each element of this list has the following structure:

```
Subject = [Semester-Field-HT-HP-DT-DP, TT, TP]
```

Prolog snippet 1.2: Modeling of the Subject entity in Prolog

Where $\boldsymbol{Semester}$, $\boldsymbol{Field}$, $\boldsymbol{HT}$, $\boldsymbol{HP}$, $\boldsymbol{DT}$ and $\boldsymbol{DP}$ are initially known values as previously described and $\boldsymbol{TT}$ and $\boldsymbol{TP}$ represent the decision variables for each subject. These are both lists, their length matches the number of teachers so, in essence, each list element matches a teacher and each list element value, say $\boldsymbol{TT_i}$, represents the number of hours teacher $\boldsymbol{Teacher_i}$ lectures for that subject. Zero values would therefore mean the $\boldsymbol{Teacher_i}$ would not be giving any theoretical hours for this subject. The exact same principle is applied to $\boldsymbol{TP}$, $\boldsymbol{TP_i}$ and $\boldsymbol{Teachers_i}$. The domain for these decision variables is as follows:

$$\boldsymbol{0 \leq TT_i * DT \leq HT}$$

$$0 \leq \boldsymbol{TP_i} * \boldsymbol{DP} \leq \boldsymbol{HP}$$

These stepped intervals are achieved in *Prolog* using our dedicate predicates: ***generateFdset*** and ***domainFdset***.

4.2 Constraints

According to the problem formulation, the following restrictions apply (using *clp(FD)* notation):

Teachers
The following constraints are applied recursively to each element of ***Teachers***, and aim only at restricting $\boldsymbol{HS1}$ and $\boldsymbol{HS2}$:

- The teacher's preference has to be respected: `Diff #= HS1 - HS2`
- The average weekly hours cannot be surpassed: `2 * Avg #>= HS1 + HS2`
- (optional) A teacher cannot have 0 classes assigned: `HS1 #> 0 #\/ HS2 \#> 0`

Subjects

The following constraints are applied recursively to each element of ***Subjects***:

- The number of practical hours is greater than the theoretical: `HP #> HT`
- The sum of $\boldsymbol{TT}$ (the theoretical hours given by teachers) must match $\boldsymbol{HT}$: `sum(TT, #=, HT)`
- The sum of $\boldsymbol{TP}$ (the practical hours given by teachers) must match $\boldsymbol{HP}$: `sum(TP, #=, HP)`
- Consider a logical matrix:

$$\boldsymbol{B} = \begin{bmatrix} \boldsymbol{b_{11}} & \boldsymbol{b_{12}} & \dots & \boldsymbol{b_{1t}} \\ \boldsymbol{b_{21}} & \boldsymbol{b_{22}} & \dots & \boldsymbol{b_{2t}} \\ \vdots & \vdots & \ddots & \vdots \\ \boldsymbol{b_{f1}} & \boldsymbol{b_{f2}} & \dots & \boldsymbol{b_{ft}} \end{bmatrix}$$

Where $\boldsymbol{f}$ is the field count and $\boldsymbol{t}$ the teachers count
also $\boldsymbol{b_{ij}}$ is the **complemented** value of the following statement:
$\boldsymbol{Teacher_j}$'s field is $\boldsymbol{i}$

With this matrix we can apply some very powerful restrictions to a given subject, by using the corresponding line of $\boldsymbol{B}$, $\boldsymbol{B_i}$:

- To force that only teachers of this field teach its theoretical lessons:
  `scalar_product(Bi, TT, #= , 0)`
- To take a first step in measuring the number of teachers (from a different field) that teach practical lessons:
  `scalar_product(Bi, TP, #= , CurrentCount)`
  (***CurrentCount*** will subsequently be used in section 4.3, in the context of solution optimization)

Teachers and Subjects
Additionally, a predicate used for more high level constraints that affects decision variables on both $\boldsymbol{Teachers}$ and $\boldsymbol{Subjects}$ was developed.
$\boldsymbol{restrictSumBySemester}$ ensures that, for each semester individually, the sum of all the classes taught in that semester, in each index $_i$ of $\boldsymbol{TT}$ and $\boldsymbol{TP}$ - i.e. in each teacher - matches $\boldsymbol{Teachers_i.HS1}$.
This is achieved by creating a matrix $\boldsymbol{M}$ where every subject produces two lines, corresponding to their own $\boldsymbol{TT}$ and $\boldsymbol{TP}$. This matrix is then reduced to a single list $\boldsymbol{L}$ where:

$$\boldsymbol{L_j = \sum_{i=0}^{\|M\|} M_{ij}}$$

A list is then obtained with all $\boldsymbol{HS1}$ values of $\boldsymbol{Teachers}$ and then the predicate `restrictEqualLists(L1, L2)` is invoked to force `L1_i #= L2_i`.

4.3 Evaluation Function

Since, in essence, this is an optimization problem, an optimization goal is of great importance [5], as such a few remarks are required so as to specify what is it that we intend to optimize:

1. Minimize the biases of the real vs expected hours a teacher can teach, among all teachers. We formalized this sentence into the minimization of the mean absolute percentage error, which can be reduced to a mean percentage error $\boldsymbol{MPE}$, since $\boldsymbol{Y_i - \hat{Y}_i \geq 0}$:

$$\boldsymbol{MPE = \frac{100\%}{n} \sum_{i=1}^{n} \frac{Y_i - \hat{Y}_i}{Y_i}}$$

   Where $\boldsymbol{n}$ is the number of teachers

$$\boldsymbol{Y_i = 2 * Teachers_i.Avg}$$

$$\boldsymbol{\hat{Y}_i = Teachers_i.HS1 + Teachers_i.HS2}$$

   However, due to the fact that we are dealing with finite domains, $\boldsymbol{MPE}$ could not reflect the error in a significant way, as all the error information would be lost in the error quotient. In order to make this a viable metric, the formula was adapted by introducing a multiplier, **1.000.000**:

$$\boldsymbol{MPE_{ratio} = \frac{1}{n} \sum_{i=1}^{n} \frac{1000000(Y_i - \hat{Y}_i)}{Y_i}}$$

2. Minimize the number of teachers that give practical lessons from fields that are not their own field of expertise. To achieve this an accumulated value of the $\boldsymbol{CurrentCount}$, $\boldsymbol{ACC}$, was used as described in Section 4.2, this will therefore hold the number of classes in these circumstances, by applying a significantly equal multiplier as in the previous optimization goal, **1.000.000**, and reflecting the value into a ratio $\boldsymbol{RP}$, it can be used as a minimization goal as well:

$$\boldsymbol{RP = \frac{1000000 * ACC}{CPL}}$$

   Where $\boldsymbol{CPL}$ is the total number of practical lessons.

Since there are two goal functions a weight factor was decided for each in order to create a unifying heuristic as follows:

$$\boldsymbol{Heuristic = 0.8 * MPE_{ratio} + 0.2 * CPL}$$

The weight of each variable should be viewed as a mere example, since investigating the correct weight of each factor falls out of the scope of the present paper. The core *Prolog* predicate that handles the above is $\boldsymbol{getHeuristicValue}$. The value of $\boldsymbol{Heuristic}$ will later be used in the search strategy as the value of the $\boldsymbol{minimize}$ labeling option.

4.4 Search Strategy

Although a good domain specification and strong constraints are the basis to work with for this kind of approach, there is also an important step to consider, which can have great impact on the efficiency of the final solver - the search strategy. This strategy can be divided into categories:

Variable Selection Strategy
The default for this setting is *leftmost*, that means the next variable to label is chosen as the leftmost one in the list of variables to label.

In our case, the variables to label, $\boldsymbol{Vars}$, are the result of appending all the unlabeled teacher variables (a list of each $\boldsymbol{HS1}$ followed by each $\boldsymbol{HS2}$ for every teacher, in order) to the unlabeled subject variables (a list of all the variables in each $\boldsymbol{TT}$ followed by the variables in each $\boldsymbol{TP}$, for every subject).

As such, choosing the leftmost variable proves to be a meaningless option. From the remaining options for this predicate, *min* and *max* were experimented on but, as expected, they yielded no significant improvements to the default option, since the $\boldsymbol{Heurisic}$ we are trying to minimize does not reflect a lower or higher value in a direct relation to the variables having small or large labeled values.

Tests were also deployed for the most recent selection strategies - *anti_first_fail*, *occurrence*, *max_regret* - but these proved to be quite inefficient for large problems and some failed to produce a single solution in twice the time that, for instance, *ffc* takes to produce a reasonable solution.

For the remaining two options, *ff* and *ffc*, we denoted efficiency improvements from the previous flags.

Value Order Strategy
As for the two possibilities in value order, *up* (the default) and *down*, we delved into the context of the problem and came up with two intuition generated ideas:

1. If *up* is used, presumably it results in assigning less hours at a time which means the conflicts are expected to appear at deeper levels of the search tree, reducing the number of possible backtracks before a solution is found;
2. If *down* is used, presumably it results in assigning many hours to each teacher and therefore the conflicts can be detected at an upper level of the search tree, quickly excluding larger subtrees.

The fact is that empirically, *down*'s overall behavior is to fail a significantly smaller amount of times than *up*, when the timeout variable is set to 15s see Figure 2.

Branching Strategy
As the problem at hand consists of minimizing a cost function, we value pruning at the earliest time possible, given this, domain splitting appears to be the sensible approach [4] [10], which can be achieved by using *bisect* as the branching strategy instead of *step* or *enum*

4.5 Solution Presentation

In order to visualize a solution in a readable way, we isolated a few predicates in a Prolog file (*display.pl*) that represents the final hour allocation in two tables, one for the teachers and another for the subjects. Figure 1 represent an example of a solution output (The Heuristic value is converted into a percentage so as to improve readability).

```
Teachers Distribution (8)
    ********************************
    * ID  FIELD AVG  DIFF  HS1  HS2 *
    *  1    1    9     0    9    9  *
    *  2    1    8     0    8    8  *
    *  3    1    7     0    6    6  *
    *  4    1    9     0    8    8  *
    *  5    1    9     0    4    4  *
    *  6    1    9     0    2    2  *
    *  7    1    9     0    2    2  *
    *  8    1    8     0    6    6  *
    ********************************

Subjects Distribution (8)
    ************************************************************************************************
    * ID  Field  Semester  HT  DT  HP  DP   Theoretical(Tid-Hours)      Practical(Tid-Hours)     *
    *  1    1         1     4   2    9   3           [1-4]                  [1-3,2-6]           *
    *  2    1         2     0   0   13   1            []                  [1-1,4-6,8-6]         *
    *  3    1         2     4   4    8   2           [1-4]                  [1-4,2-4]           *
    *  4    1         1     0   0   12   2            []                [1-2,2-2,3-4,4-4]       *
    *  5    1         1     2   1    8   2          [3-2]                   [4-4,5-4]           *
    *  6    1         2     0   0   10   2            []                  [2-4,3-2,5-4]         *
    *  7    1         2     4   1    6   2        [6-2,7-2]                  [3-4,4-2]           *
    *  8    1         1     0   0   10   2            []                  [6-2,7-2,8-6]         *
    ************************************************************************************************
For the next 3 values "0" represents a perfect score for the given area:
Heuristic: 26.15%
Failed Hours: 32.69%
#Practical Teacher From Other Field: 0
```

Fig. 1: Example of a solution output

In order to invoke the solver the following options exist (by default ***Timeout*** = 15000 and ***Debug*** = false, ***Timeout*** is the number of milliseconds to use for the time_out and ***Debug*** specifies the level of verbosity):

1. `init(Subjects, Teachers)` Debug = default, Timeout = default
2. `init(Subjects, Teachers, true)` Debug = true, Timeout = default
3. `init(Subjects, Teachers, false)` Debug = false, Timeout = default
4. `init(Subjects, Teachers, T)` Debug = default, Timeout = T
5. `init(Subjects, Teachers, true, T)` Debug = true, Timeout = T
6. `init(Subjects, Teachers, false, T)` Debug = false, Timeout = T

5 Results

The following results were measured by using an automatic process that uses our generator algorithm to create problems of known and increasing complexity and that submits them

to our *clp(FD)* solver saving the results in a *.csv* file that was then treated for relevant metrics. For large problems, that cannot be optimally solved in useful time, a timeout of 15s was introduced, when the main comparison goal was to see the overall performance of the labeling options.
Figure 2 reveals the results of a stress test done at different labeling options. Noting that all the options except for *leftmost_up* use the *bisect* branching option as discussed in section 4.4, *leftmost_up* uses all the default options and therefore *step* as the branching strategy. This figure reveals that for a total of 336 tests of increasingly more complex problems (1 to 8 rounds of the test generator, which produce values from 2 subject and 2 teachers to 16 subjects and 16 or more teachers) the usage of the *down* flag, for the 15s timeout, is quite a better than *up* flag, and that the combination of *leftmost* and *step* behaves worse than any combination of *ff* or *ffc* with *bisect*.

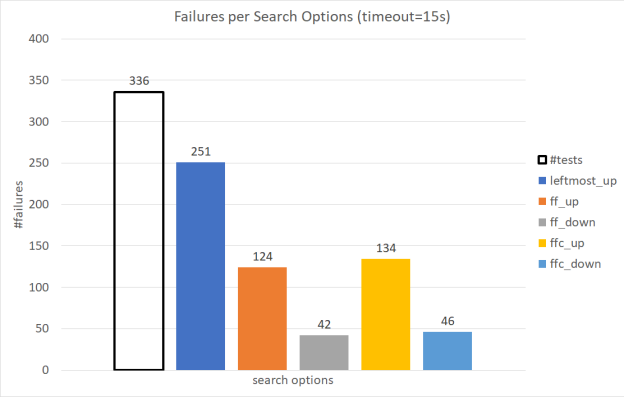

Fig. 2: Test suite result - failures per labeling options (out of 336) for 15s - increasing complexity

Figure 3 shows the approximately linear behavior of the solver in what concerns constraint application. We should denote that a given round can generate problems within a fixed interval of complexity, which means that the observable discrepancy in between the number of constraints across different number of rounds to a linear function is expected to exist and should be minimized as the number of repetitions increases.

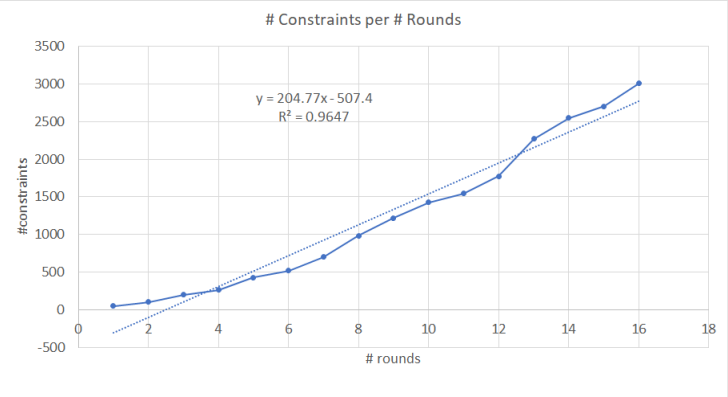

Fig. 3: Progress of constraints as the complexity of the problem increases

Figure 4 shows the same search options as used in Figure 2 but from a backtrack count perspective. In it we can observe that for smaller test cases, the default options reveal a smaller number of backtracks, but as complexity increases so does the backtracks ratio of the default options versus any of the remaining options. We should denote that for complexity greater or equal to 3 the 15s timeout kicks in and that the default option cannot produce a single viable option for the problem, whereas the remaining flags can provide at least (but never just) one solution out of 20 attempts. This test could be applied for longer timeout periods so as to better ascertain the underlying behavior.

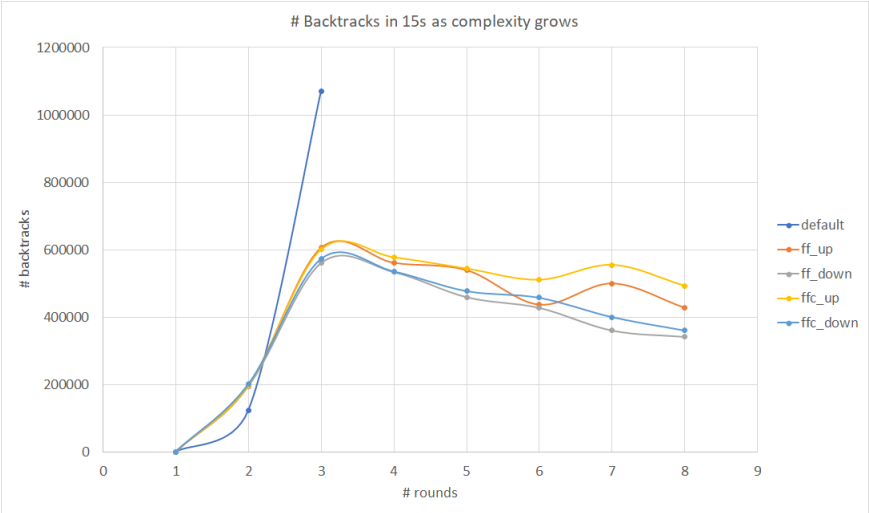

Fig. 4: Evolution of the number of backtracks with the increasing complexity, for different search options - timeout 15s

6 Conclusion

The present project has brought interesting constraint programming concepts together with imperative approaches, as used in the test case generator. It has yielded some interesting results for the *Teacher Hour Allocation for University Subjects* problem and fortunately enough it revealed many potential improvements that were not previously envisioned, namely:

- The presented solver - and subsequent improvements - should be subjected to a good debugging tool for constraint programming in *SICStus Prolog*, such as the *Finite domain DeBuGer* [11] which bring about faster and more efficient development.
- Another improvement on the proposed solution arose at an advanced level of development and was not, therefore, implemented. Yet, a wish for a test of its impact on the solver does exist. That is, to model the teacher variables by separating them into $\boldsymbol{HS1T}$ and $\boldsymbol{HS1P}$, representing a subdivision of $\boldsymbol{HS1}$ for the theoretical and practical lessons taught by a teacher (the same goes for $\boldsymbol{HS2}$) as this would allow the application of new constraints that target theoretical and practical hours, for teachers, individually.
- Moreover, a comparison between the constraints described in Section 4.2 that affect the allocation of a teacher to a class and the *clp(FD)* library's *cumulatives* predicate is desired. Mainly because this could prove a novel and useful comparison to the aforementioned predicate when it is not used in its fullness. This is because a view of teachers as Machines and of teachers as Tasks represents only a partial equality to the *cumulatives* predicate, as the time constraints this predicate imposes would not apply to our problem, only the constraints concerning resource (teacher hours) allocation, and so this possible optimization must be tested.
- On top of the results presented, a better tuning of the search flags is desired that could be achieved through the application of long-lasting tests (function that could easily be achieved using our Python3 testing tools and a few computing days' time). This, we believe, would do a lot into fine-tuning the solver performance.
- Another improvement that revealed itself at a late phase of the present paper is the simple fact of grouping similar teachers (same $\boldsymbol{Field}$ and $\boldsymbol{Avg}$) together and consider them as indistinguishable. By doing so, and with some adapted constraints, the initial complexity of the problem could be reduced and the existence of symmetries avoided, as is usually the goal in this class of problems.

Overall, the project has proved to be an interesting approach to delve deeper into the ocean that constraint logic programming is. We saw those rare creatures that hide way underneath the surface, such as the impact a labeling option can have on the efficiency of a search. We also had the opportunity to witness firsthand an overall view of the power and simplicity that this declarative and intuitive approach can bring to typically nondeterministic polynomial time problems, those white sharks also known as NP.

References

1. Esraa A. Abdelhalim and Ghada A. El Khayat. A Utilization-based Genetic Algorithm for Solving the University Timetabling Problem (UGA). *Alexandria Engineering Journal*, 55(2):1395–1409, 2016.
2. Roman Barták. Effective modeling with constraints. *Applications of Declarative Programming and ...*, pages 149–165, 2005.
3. John Charnley and Simon Colton. Expressing General Problems as CSPs.
4. John G Cleary. Proving the existence of solutions in logical arithmetic. 1993.
5. A. Colorni, M. Dorigo, F. Maffioli, V. Maniezzo, G. Righini, and M. Trubian. Heuristics from Nature for Hard Combinatorial Optimization Problems. *International Transactions in Operational Research*, 3(1):1–21, 1996.
6. D. de Werra. An introduction to timetabling. *European Journal of Operational Research*, 19(2):151–162, 1985.
7. Andreas Drexl and Frank Salewski. Distribution requirements and compactness constraints in school timetabling. *European Journal of Operational Research*, 102(1):193–214, 1997.
8. Rainer Kalisch. *Project Scheduling under Resource Constraints - Efficient Heuristics for Several Problem Classes*. 1995.
9. Olivier Lhomme. Consistency techniques for numeric CSPs. *Ijcai*, pages 232–238, 1993.
10. David L. Poole and Alan K. Mackworth. *Artificial intelligence: Foundations of computational agents*. 2010.
11. Dávid Hanák Szeredi and Tamás. FDBG, the CLP(FD) Debugger Library of SICStus Prolog.
12. L. Zhang and S.K. Lau. Constructing university timetable using constraint satisfaction programming approach. *International Conference on Computational Intelligence for Modelling, Control and Automation and International Conference on Intelligent Agents, Web Technologies and Internet Commerce (CIMCA-IAWTIC'06)*, 2:55–60, 2005.