Facing the Curriculum-Based Course Timetabling (CB-CTT) problem using a SMT approach.

Adrián García and Estevo Cordal

February 4, 2021

1 Motivation

The CB-CTT is a widely-known problem, many schools, universities and other services have to yearly deal with how to organize their schedules for students and employees. Hence, it is a big challenge for us to get a closer solution to this matter. We will use the knowledge acquired during Formal Methods in Software Development course for this purpose, trying to solve it using a SMT solver, specifically the Microsoft Z3 Solver [1].

2 Problem Definition

1. The CB-CTT problem consists of finding the best weekly schedule of university lectures to rooms and time periods.
2. For this problem, a set of constraints [2] is given by its own nature, divided into two subgroups:
   - A set of **hard** constraints, that are mandatory to be met.
     - **A teacher can only teach a course at a time**: For obvious reasons, each teacher can only give one lecture simultaneously.
     - **A room can't hold 2 lectures simultaneously**: In the same room there can't be two lectures for the same time period and day.
     - **Courses of same year can't be scheduled in parallel**: We must guarantee that subjects corresponding to a curriculum year won't have any lectures at the same time, therefore, all students can access each of them.
     - **Each class has to have four weekly lectures**: The number of weekly lectures per subject has to be exactly 4 hours.
   - A set of **soft** constraints that should be met if possible:
     - The **capacity** of the room should be at least equal to the number of enrolled students for the lecture which will take place there.
     - There should not be **time spaces** between same year classes for each day, i.e. all classes should go one after the other (in same curriculum year).
     - **Room stability** should be ensured, which means that for a given subject, its classes may be always in the same room.
     - We would like to ensure that classes that are divided into **subgroups have the same time period** assigned, although in different rooms.

3. To be able to use the Z3 solver, we have to create different variables, we opted for an Integer approach, although a Boolean one would also be appropriate. We encoded our variables following this template:

- [**Year**].[**Course**].[**Teacher**].[**Room**].[**Time-Period**].[**Week-Day**]
  - **Year**: It references the year in which the course takes place, it could range from 1 to 4 (2 semesters/year) but we only used first semester subject between years 1 and 2 for our schedule.
  - **Course**: It represents the ID of each subject, which is linked to a Python dictionary that contains more details about it (Subject official name, Teacher name and number of enrolled students)
  - **Teacher**: This value is taken from the previous dictionary directly by our code, it compares the teacher name in the dictionary and gives a unique ID for each teacher.

Courses Dictionary				
Course ID	Curriculum Year	Course Name	Teacher Name	Enrolled Students
1	1	Discrete Maths	Gilberto Perez	78
2	1	Calculus	Jose Rodriguez	80
3	2	Electronics Technology	Julio Bregains	124
4	2	Electronics Technology	Jose Naya	130
5	1	Programming I	Cristian Munteanu	92
6	2	Programming II	Carlos Gomez	58
7	2	Computer Fundamentals	Carlos Regueiro	24
8	1	Basic Finances	Ramon Minones	38
9	1	Computer Basics	Guillermo Roca	72
10	2	Algebra	Cristina Costoya	130
11	2	Statistics	Patricia Gonzalez	93

  - **Room**: It represents another dictionary, so each value is also unique, when accessing the dictionary we can get info about each room (Room name, maximum capacity)
    - **ID 1**. Room A1.0 - Capacity: 80
    - **ID 2**. Room A1.1 - Capacity: 25
    - **ID 3**. Room A1.2 - Capacity: 140
    - **ID 4**. Room A2.0 - Capacity: 60
  - **Time-Period**: Ranging from 1 to 9, it represents the starting and ending time for each lecture.
    - **ID 1**. From: 08:00 to 09:00
    - **ID 2**. From: 09:00 to 10:00
    - **ID 3**. From: 10:00 to 11:00
    - **ID 4**. From: 11:00 to 12:00
    - **ID 5**. From: 12:00 to 13:00
    - **ID 6**. From: 13:00 to 14:00
    - **ID 7**. From: 15:00 to 16:00
    - **ID 8**. From: 16:00 to 17:00
    - **ID 9**. From: 17:00 to 18:00
  - **Week-Day**: Starting from 1 up to 5, it means the day of the week that the lecture will take place in.
    - **ID 1**. Monday
    - **ID 2**. Tuesday
    - **ID 3**. Wednesday
    - **ID 4**. Thursday
    - **ID 5**. Friday

- As an example, variable **1.2.4.1.3.4** represents an Int variable for a Semester **1** subject, in this case for Course **2** given by Teacher **4**. The lecture will be at Room **1** for Time period **3** and Day **4**. If this variable is equal to 1, it means that the given course will take place in that period. Otherwise, this combination will not occur.

4. - Having all the initial constraints clear and our variables created, we proceed translating them into code to be solved by Z3 solver. The structure in Python for all of the constraints is pretty similar, the premise is that the variables represent each possible state in the timetable, and we want to give the value 1 to those that would be correct following our constraints (or 0 for the ones that don't, for that matter) but we can't select each one by one, that's where Z3 comes in. We traverse our data dictionaries and select the variable codes to form a whole variable. After that, we add them to a list to give it as an assertion to our solver. Here's an example:

```
def only_one_class_per_room(solver):
    for i in range(1, len(rooms) + 1):
        for j in range(1, len(week_days) + 1):
            for k in range(1, len(time_periods) + 1):
                xor_list = []
                for m in range(1, len(courses) + 1):
                    data = courses.get(str(m)).split(" ")
                    s = data[0]
                    for l in range(1, len(teachers) + 1):
                        if data[2] != list(teachers.keys())[list(teachers.
    values()).index(str(l))]:
                            continue
                        var = str(s) + '.' + str(m) + '.' + str(l) + '.' + str(
    i) + '.' + str(k) + '.' + str(j)
                        pos = var_list.index(var)
                        xor_list.append(bools[pos])
                solver.add(Sum(xor_list) <= 1))
```

Example of one of the constraints added with this function: $1.1.1.1.1.1 + 1.2.4.1.1.1 + 2.3.9.1.1.1 + 2.4.11.1.1.1 + 1.5.2.1.1.1 + 2.6.10.1.1.1 + 2.7.7.1.1.1 + 1.8.3.1.1.1 + 1.9.5.1.1.1 + 2.10.6.1.1.1 + 2.11.13.1.1.1 \leq 1$
Total number of constraints added by this function: 1980 assertions

3 Solution

1. Obtaining a solution for this problem is expensive in terms of time as it increases exponentially depending on courses, time periods, rooms and days. It is due to the fact that thousands of constraints are being added to the Z3 solver, a lot of them as soft constraints so the solver tries to optimize them, which takes a lot of time as it constantly recalculates everything trying to maximize the number of soft constraints that are validated.

2. At first we tried to solve this problem with the data of our faculty at our home university in Spain (Facultade de Informatica de A Coruña. Universidade da Coruña) but we didn't manage to get a solution, we stopped the execution of the program after 7.000 seconds. We then realized that we had to work with less data, so we used only the courses for the first semester for years 1 and 2, only 4 rooms and 9 time periods. Here you can see the total execution time for this data:

```
sat
Time to check sat:  1070.0569751262665  seconds.
Total time including assertions creation:  1072.13316822052  seconds.

Process finished with exit code 0
```

The solver outputs the following model: 1.1.1.1.2.2 - 1.1.1.1.6.2 - 1.1.1.1.6.3 - 1.1.1.1.8.3 - 1.2.4.1.2.4 - 1.2.4.1.3.1 - 1.2.4.1.3.4 - 1.2.4.1.4.2 - 2.3.9.1.2.3 - 2.3.9.1.6.1 - 2.3.9.1.8.2 - 2.3.9.1.8.4 - 2.4.11.3.1.1 - 2.4.11.3.1.5 - 2.4.11.3.4.1 - 2.4.11.3.4.4 - 1.5.2.3.2.5 - 1.5.2.3.6.1 - 1.5.2.3.6.5 - 1.5.2.3.8.4 - 2.6.10.4.4.2 - 2.6.10.4.4.5 - 2.6.10.4.7.2 - 2.6.10.4.9.4 - 2.7.7.4.3.2 - 2.7.7.4.6.3 - 2.7.7.4.8.1 - 2.7.7.4.9.1 - 1.8.3.1.1.1 - 1.8.3.1.3.2 - 1.8.3.1.4.5 - 1.8.3.1.7.2 - 1.9.5.1.4.1 - 1.9.5.1.7.3 - 1.9.5.1.7.5 - 1.9.5.1.9.3 - 2.10.6.3.1.2 - 2.10.6.3.6.2 - 2.10.6.3.7.4 - 2.10.6.3.8.3 - 2.11.13.3.3.3 - 2.11.13.3.3.4 - 2.11.13.3.7.1 - 2.11.13.3.9.2

4 Examples

Schedule for the first semester year 1:

	Monday	Tuesday	Wednesday	Thursday	Friday
08:00 - 09:00	Basic Finances A1.0 Ramon Minones				
09:00 - 10:00		Discrete Maths A1.0 Gilberto Perez		Calculus A1.0 Jose Rodriguez	Programming I A1.2 Cristian Munteanu
10:00 - 11:00	Calculus A1.0 Jose Rodriguez	Basic Finances A1.0 Ramon Minones			
11:00 - 12:00	Computer Basics A1.0 Guillermo Roca	Calculus A1.0 Jose Rodriguez			Basic Finances A1.0 Ramon Minones
12:00 - 13:00					
13:00 - 14:00	Programming I A1.2 Cristian Munteanu	Discrete Maths A1.0 Gilberto Perez	Discrete Maths A1.0 Gilberto Perez		Programming I A1.2 Cristian Munteanu
14:00 - 15:00					
15:00 - 16:00		Basic Finances A1.0 Ramon Minones	Computer Basics A1.0 Guillermo Roca		Computer Basics A1.0 Guillermo Roca
16:00 - 17:00			Discrete Maths A1.0 Gilberto Perez	Programming I A1.2 Cristian Munteanu	
17:00 - 18:00			Computer Basics A1.0 Guillermo Roca		

As we can see, there are some free periods between courses which means that the schedule is not optimal, the main reason is that we had to run it with only 4 rooms available to avoid eternal execution times. Because of that, some courses can't go one after the other since subjects from first semester year 2 also have to fit simultaneously with the ones seen in this schedule.

Schedule for the first semester year 2:

	Monday	Tuesday	Wednesday	Thursday	Friday
08:00 - 09:00		Algebra A1.2 Cristina Costoya			
09:00 - 10:00			Electronics T. A1.0 J.Bregains A1.2 J.Naya		
10:00 - 11:00		Computer F. A2.0 Carlos Regueiro	Statistics A1.2 Patricia Gonzalez	Statistics A1.2 Patricia Gonzalez	
11:00 - 12:00		Programming II A2.0 Carlos Gomez			Programming II A2.0 Carlos Gomez
12:00 - 13:00					
13:00 - 14:00	Electronics T. A1.0 J.Bregains A1.2 J.Naya	Algebra A1.2 Cristina Costoya	Computer F. A2.0 Carlos Regueiro		
14:00 - 15:00					
15:00 - 16:00	Statistics A1.2 Patricia Gonzalez	Programming II A2.0 Carlos Gomez		Algebra A1.2 Cristina Costoya	
16:00 - 17:00	Computer F.	Electronics T. A1.0 J.Bregains A1.2 J.Naya	Algebra A1.2 Cristina Costoya	Electronics T. A1.0 J.Bregains A1.2 J.Naya	
17:00 - 18:00	A2.0 Carlos Regueiro	Statistics A1.2 Patricia Gonzalez		Programming II A2.0 Carlos Gomez	

Overall, the schedule is quite good and fulfills at its best all the constraints given to the solver.

5 Tools used to obtain the solution

1. **Python.** We used it as the programming language to create all the functions and methods.
2. **Z3 Solver.** We used it to solve the CB-CTT following a SMT approach.

References

[1] Microsoft Corporation. Z3 solver, 2012. Licensed by Massachusetts Institute of Technology.

[2] Alexandre Lemos, Francisco S. Melo, Pedro T. Monteiro, and Inês Lynce. Room usage optimization in timetabling: A case study at universidade de lisboa. *Operations Research Perspectives*, 6:100092, 2019.