Department of Mathematics and Computer Science
University of Southern Denmark, Odense

October 26, 2020
Marco Chiarandini

DM877 - Constraint Programming

Final Obligatory Assignment, Autumn 2020

Deadline: Monday, January 11, 2021 at noon.

Preamble

- This is the *final* assignment on constraint programming. The assignment will be graded with internal censorship and the grade will be your final assessment for the course.
- The assignment has to be carried out individually. Communication for feedback and inspiration with peers is allowed but it should be kept to a minimum. In case, you are welcome to ask to the teacher.
- The submission is electronic in BlackBoard via SDU Assignment.
- You have to submit:
  - a PDF document containing max 10 pages of description of the model and the results you have obtained on the data sets provided
  - source code of your implementation of the model in MiniZinc.
- The data sets associated with the assignment is available at:

  https://imada.sdu.dk/~marco/DM877/assets/exams.tgz

Problem Description

The task of this assignment is using Constraint Programming to schedule the exams at the Faculty of Science at SDU for the ordinary session of January 2021.
Let E be the set of exams for courses that need to have a date scheduled. Let D be the set of integer numbers representing the days that are available for scheduling exams. The set D may contain holes due to weekends and holidays. Exams that for some reason cannot have overlapping schedules are called conflicting exams. In particular, exams with the same teacher or censor and exams that belong to the same study program are considered conflicting.
An *exam schedule* is an assignment of exams to days for the exam, that is, it is a function $\sigma : E \to 2^D$. An exam schedule is *feasible* if it satisfies the following constraints:

1. Each exam in input is scheduled to start and finish in the days available.
2. Each exam is assigned a number of days, $|\sigma(e)|$, equal to the number of required days to carry out the exam.
3. Exams with exam duration more than one day, receive consecutive days. No holes due to weekend or holidays are allowed within an exam schedule.
4. Exams with preassigned dates must respect those dates.
5. Exams that are *joined* must have schedules starting on the same day.[1]
6. Exams that are *conflicting* must be scheduled in days with empty intersection.

[1]The durations for both should be the sum of the durations of the two exams but this is less important.

7. The total number of students having *written* exams in a day must be smaller than `MAX_SEATS_PER_DAY`. This includes exams with predefined dates.
8. The total number of oral exams in a day must be smaller than `MAX_ROOMS_PER_DAY` in order to ensure that a room to host them can be found. This includes exams with predefined dates.

If a feasible schedule that satisfies all the above constraints is found, then one should try to maximize the distance between the starting times of exams that share students. This optimization task can be achieved in different ways, for example, by introducing a new constraint that enforces a distance of R days between exams that share students and then maximizing R and/or minimizing the number of students violating this constraints or by devising a penalty scheme that penalizes exams sharing students that are scheduled too close.
If a feasible schedule cannot be found then one must relax the constraints on the conflicting courses and treat them as those sharing students but with higher priority in avoiding overlaps. The parameter `WEIGHT_PROGRAMS` indicates the relative importance between avoiding overlaps for conflicting courses and avoiding overlaps for courses that only share students.

Input Data

You are given a starting package containing data on the exams for E20 at the Faculty of Science, SDU and some Python code to read the data, carry out a few calculations, and output it in the format you wish.

Data about the days available for scheduling and the parameters `MAX_SEATS_PER_DAY`, `MAX_ROOMS_PER_DAY` and `WEIGHT_PROGRAMS` is provided in a json file `config.json` (see Fig. 1). The field `DAYS` contains a list of days with weekends and holidays already removed. The days are given as a number from `DAYONE`. You do NOT need to convert them back in dates. The other fields in the input data are not relevant for this assignment.

Data about the exams is provided in a json file consisting of a dictionary whose elements are the exams $e \in E$. See Figure 2. The key of each exam $e \in E$ is the STADS identifier. Then, for each entry corresponding to an exam the following information is provided (not all the information is relevant for this assignment):

- `EKA` the STADS code of the exam. The same as the key.
- `STADS_ID` the STADS code of the course t which the exam belong
- `NAT_code` the corresponding code of the course at the Faculty of Natural Sciences. This might not be unique as a course might have both a written exam and an oral exam.
- `title` the title of the course
- `ECTS` the number of ECTS for the course
- `Prøveform` the form of the exam: written or oral
- `stype` a shortcut for the form of exam: 'm' for oral and 's' for written
- `Administrationsenhed` the administering unit
- `institute` the administering institute
- `ECTS/prøve` the ECTS per exam component
- `Eksamensperiode` when the exam has to be offered: in the ordinary session and in the reexam session
- `students` a list of students registered to the exam
- `nstds` the total number of students registered
- `rdays` duration r_e of the exam expressed in number of required days. A written exam requires one day while the number of days of an oral exam is the number of students divided by 16, the number of students that can be examined orally in one day.
- `schedule` if the exam has already been scheduled then information about the schedule is provided here as a list of days given by the pair number of day, date.
- `Faste datoer` preassigned dates the same as the previous field, it can be ignored.
- `joined` a list of exams that must be scheduled jointly, that is starting on the same days

```
{
    "DISTANCE": 3,
    "SEMESTER": "e2020",
    "SESSION": 0,
    "MAX_SEATS_PER_DAY": 300,
    "MAX_ROOMS_PER_DAY": 20,
    "MAX_STUDENTS_PER_DAY": 16,
    "WEIGHT_PROGRAMS": 100,
    "YEAR": 2021,
    "MAX_EXAMS": 15,
    "MAX_ECTS": 40,
    "FIRST_DAY": [1,4],
    "LAST_DAY": [1,29],
    "MONTH": 1,
    "DAYS": [4,5,...,28,29],
    "DAYONE": "2021-01-01"
}
```

Figure 1: An example of input data for the days and other parameters

- `conflicting` a list of exams that must necessarily be scheduled in different days.

An instance of the exam schedule problem consists of two files: `config.json` and a file containing details about the exams to schedule. Specifically, you are given exams composing five different instances of the problem with different size and difficulty. See Figure 1.

Exams in biologi	8
Exams in bmb	17
Exams in imada	41
Exams in fkf	45
Exams in all	123

Table 1: The input data divided into 5 instances.

The Python code calculates a matrix with the number of shared students between exams that can turn out relevant for the assignment.

Your Tasks

For an excellent performance you have to give account of the tasks below in a written report of a most 10 pages (using the template provided in Assignment 1):

1. design a CP model to find exam schedules according to the constraints and criteria described above
2. address symmetry issues, if any
3. implement the model in MiniZinc
4. test alternative variable-value heuristics
5. test the contribution of random restarts

Run all computational experiments for a maximum time of 20 minutes. Report the results in tables containing at least the following columns:

- name of the instance (biologi, bmb, imada, fkf, all)
- time (in sec) when a solution was found or when the program was stopped
- number of nodes explored
- whether a solution was found

```
{
 "N300003102": {
        "STADS_ID": "N300003101",
        "NAT_code": "MM531",
        "title": "Differentialligninger II",
        "ECTS": 5,
        "Administrationsenhed": "IMADA",
        "EKA": "N300003102",
        "Prøveform": "Mundtlig prøve",
        "ECTS/prøve": 5.0,
        "Eksamensperiode": "Ordinær/reeksamen",
        "Faste datoer": null,
        "stype": "m",
        "institute": "imada",
        "students": [
            "xxxxxxx",
            "yyyyyyy",
            "zzzzzzz",
            "wwwwwww",
        ],
        "nstds": 4,
        "rdays": 1,
        "joined": [
            "N310004102"
        ],
        "conflicting:" []
    },
...
}
```

Figure 2: An example of input data for the exams

- if a solution was not found how many constraints the best solution found was violating (or in case you have chosen to use the penalty scheme, the best penalty value reached).

If you have chosen the approach with the constraint on R then you can report the results for different values of R on the same instance, in which case R should constitute another column.