Problem 3: Schedule FAU Lectures

AI1SysProj 2021

Topic:	Constraint Satisfaction
Due on:	February 1, 2021
Version from:	January 18, 2022

1 Task Summary

Use the constraint solver Minion to schedule lectures and tutorials for the technical faculty at FAU, based on real data exported from UnivIS. The data and other relevant files are in the assignment repository at `https://gitlab.rrze.fau.de/wrv/AISysProj/ws2122/csp/assignment`.

2 Background: Lecture Scheduling at the Technical Faculty

The scheduling process at the technical faculty consists roughly of the following steps:

1. The schedule from the previous year is copied over (on UnivIS)
2. Missing/dropped lectures are added/removed and entries are updated
3. The data is fed into a scheduling tool that searches for a schedule satisfying a number of hard constraints (e.g. a professor can only hold one lecture at a time) and soft constraints (e.g. avoid evening time slots)
4. Missing and problematic entries (e.g. entries imposing too many constraints) are fixed and step 3 is repeated

You can read more about this process at [RF] (German only).

In this assignment, we will try to implement step 3 of the process. As the real scheduling problem is too complex, we will instead look at a simplified/modified problem. In particular, we will ignore many special cases and soft constraints.

3 The Data Set

Due to the Covid-19 pandemic, the current data on UnivIS is not very suitable and we will instead use data from the winter semester 2019/2020. The data is split into two files: `lectures.xml` and `rooms.xml` for data on the lectures and rooms respectively.

3.1 Lecture Data

Only the `<Lecture>` nodes are relevant for us. Listing 1 shows an example node. The important fields are

- `@key`: Attribute used as an identifier for the lecture.
- `dozs`: Lists the instructors (in this case only Prof. Kohlhase, who has the identifier `Person.tech.IMMD.pkosy.kohlha`).
- `orgunit_ens`: Lists the organizations behind the lecture.
- `studs`: Lists which students can take the lecture:
  - `stud/richt` indicates the study area (e.g. `INF-MA` for a master's students in computer science)
  - `stud/pflicht` indicates if it is an elective (`WF`), a compulsory lecture (`PF`), or a compulsory elective (`WPF`)
  - `stud/sem` lists in what semesters the lecture can be taken (as `INF-BA-V-KI` from semester 5 onwards and as `INF-MA` from semester 1 onwards). You can think of the contents of the `stud/sem` field as a list of hexadecimal digits, indicating all included semesters
- `turnout`: The expected number of students to take the lecture
- `terms`: Indicates what slots are needed. They are usually copied from the previous semester, which allows us to infer the requirements. Each `term` corresponds to one required slot, and the length of the slot can be inferred from `term/endtime` and `term/starttime`. `term/repeat` indicates how the slot should be repeated. We will ignore anything that does not start with a `w` (weekly). The `w` is followed by one digit that we ignore (presumably indicating if it is weekly/biweekly), which is then followed by a list of weekdays. For example, `w1 3,5` indicates that the slot is scheduled for weekdays `3` and `5` (Wednesday and Friday).
- `type`: Indicates the type (`vorl` means it is a lecture; there are other values for seminars (`sem`), exercises (`ue`, `v-ue`), etc.).

```
<Lecture key="Lecture.tech.IMMD.pkosy.kii">
  <dozs>
    <doz><UnivISRef type="Person" key="Person.tech.IMMD.pkosy.kohlha"/></doz>
  </dozs>
  <orgunit_ens>
    <orgunit_en>Faculty of Engineering</orgunit_en>
    <orgunit_en>Department of Computer Science</orgunit_en>
  </orgunit_ens>
  <studs>
    <stud>
      <pflicht>WPF</pflicht>
      <richt>INF-BA-V-KI</richt>
      <sem>56789ABCDEF</sem>
    </stud>
    <stud>
      <pflicht>WPF</pflicht>
      <richt>INF-MA</richt>
      <sem>123456789ABCDEF</sem>
    </stud>
  </studs>
  <turnout>100</turnout>
  <terms>
    <term>
      <endtime>13:45</endtime>
      <repeat>w1 2</repeat>
      <starttime>12:15</starttime>
    </term>
    <term>
      <endtime>13:45</endtime>
      <repeat>w1 4</repeat>
      <starttime>12:15</starttime>
    </term>
  </terms>
  <type>vorl</type>
</Lecture>
```

Listing 1: The AI 1 lecture in `lectures.xml` (many nodes have been removed for conciseness).

```
<Room key="Room.tech.zentr.zentr.h16">
  <famos_Nutzungsart>510 Unterrichtsr.m.fest.Gest&#xFC;hl</famos_Nutzungsart>
  <orgunit_ens>
    <orgunit_en>Faculty of Engineering</orgunit_en>
  </orgunit_ens>
  <size>108</size>
</Room>
```

Listing 2: An example room in `rooms.xml` (many nodes have been removed for conciseness).

3.2 Room Data

Listing 2 shows an example room. The following fields are relevant for us:

- `@key`: Attribute used as an identifier for the room.
- `famos_Nutzungsart`: Describes how the room can be used. We will only look at the code (`510` in the example, which indicates a teaching room with fixed seats).
- `orgunit_ens`: Lists the organizations responsible for the room.
- `size`: The capacity of the room (i.e. how many students can be in it at the same time).

4 Minion

Minion [GJM06] is an open source solver for constraint satisfaction problems. In this assignment, the core problem should be solved by Minion (basic pre-processing and simplification are of course allowed). That means that your implementation should generate a `.minion` file representing the scheduling problem. The variable assignments generated by Minion should then be converted into the output format (Section 7). Please briefly describe how the `.minion` file corresponds to the scheduling problem.

You can download Minion at [MD]. The download also contains a manual describing how to use Minion. Use the mattermost channel for any questions about Minion.

5 Problem 1: Warm-up

This problem covers few lectures and constraints and is intended to help you get started with the data and Minion. The goal is to create a schedule in the output format (Section 7)

that satisfies all the constraints. You can check your solution with the verification script from the assignment repository.

Lectures Only include lectures (no tutorials etc.) by the department of computer science, i.e. only lectures where

1. an `orgunit_en` is `Department of Computer Science`, and
2. `type` is `vorl`.

The number of slots that have to be allocated is the number of nodes `terms/term` that have a node `repeat` whose text content starts with `w`. For this problem we assume that every slot is 90 minutes long, i.e. we ignore the `starttime` and `endtime`.

Rooms Only use the following three rooms (identified by the `key`):

1. `Room.sonste.hrsaei.audima.audima`
2. `Room.tech.zentr.zentr.h16`
3. `Room.tech.IE.LER.hr415`

Time slots For every weekday (Monday through Friday), there are 6 time slots available: 8:15 – 9:45, 10:15 – 11:45, 12:15 – 13:45, 14:15 – 15:45, 16:15 – 17:45, 18:15 – 19:45.

Constraints Hard constraints:

1. An instructor (`dozs/doz`) can only teach one lecture at a time. We assume that all listed instructors have to be be present for the lecture.
2. Each room can only have one lecture at a time.
3. The room capacity has to be big enough to hold all the students.

Soft constraints:

1. Minimize the number of evening slots (18:15 – 19:45).

As this is a soft constraint, you do not have to find an optimal solution (the verification script will accept any solution with at most 4 evening slots, even though fewer are possible). Tip: Ignore the soft constraints in the beginning because they are trickier to implement.

Goal Transform the scheduling problem into a constraint satisfaction problem `p1.minion`, which can be solved by Minion. Use the solution to create a schedule `p1.xml` in the format specified in Section 7. You can use the verification script to check your schedule (it should catch most errors).

6 Problem 2: The Actual Scheduling Problem

This problem covers significantly more lectures and more constraints. It is much closer to the way lectures are scheduled at FAU.

Slots that require scheduling Include any slot (`Lecture/terms/term`) that fulfills *all* of the following requirements:

1. there is either no room assigned, or the assigned room has one of the following `famos_Nutzungsart` codes: `523`, `513`, `521`, `510`, `511`, `512` (the other codes are used for special rooms like labs)[1]
2. the `repeat` value starts with `w` (note that a `term` with multiple days should be treated as multiple slots)
3. the lecture it belongs to
   (a) is organized by the technical faculty (an `orgunit_en` is `Faculty of Engineering`) and
   (b) has one of the following types (`type`): `tut`, `ue`, `v-ue`, `vorl` and
   (c) has `centrally_allocated` set to `1`
4. it has a `starttime` and `endtime`

If `endtime` − `starttime` > 120 min, you should assign a double slot, otherwise a single (90-minute) slot suffices.

Rooms You can use any room that

1. has an `orgunit_en` entry with value `Faculty of Engineering` and
2. has one of the following `famos_Nutzungsart` codes: `523`, `513`, `521`, `510`, `511`, `512`, and
3. has a `size` that is larger than 1

In addition, you may use the lecture halls

- `H1` (`Room.nat.dchph.zentr.h1`)
- `H11` (`Room.nat.dma.zentr.h11`)
- `CIP2b` (`Room.tech.IMMD.zentr.02151a`)[2]

Time slots For every weekday (Monday through Friday), there are 6 time slots available: 8:15 – 9:45, 10:15 – 11:45, 12:15 – 13:45, 14:15 – 15:45, 16:15 – 17:45, 18:15 – 19:45. They can be combined into the following double slots: 8:15 – 11:45, 10:15 – 13:45, 12:15 – 15:45,

[1] Note that this includes rooms which you are not allowed to use for your scheduling (see next paragraph)
[2] Technically, this room is not necessary

14:15 – 17:45, 16:15 – 19:45.

Constraints You should treat any lecture slots that are not scheduled by you (but where `repeat` starts with `w`) as **fixed** and consider them for the constraints below (e.g. if a fixed lecture uses a room at a particular time, you cannot use that room at the same time for your scheduling).

We have the following constraints:

1. An instructor (`dozs/doz`) can only teach one lecture at a time. As before, we assume that all listed instructors have to be be present for the lecture. However, you may ignore
   (a) `Person.tech.ITC.paot.frbaan_8` (who is involved in too many lectures)
   (b) any place holder person (starting with `Person.zentr.zentr.zentr.`)
   (c) any slots of a fixed lecture that are longer than 6 hours
2. Each room can only have one lecture at a time.
3. The room capacity has to be big enough to hold all the students (for lectures without a `turnout` you should assume that 20 students will show up and for lectures with a `turnout` > 500 you may assume that only 500 students will show up).
4. All slots that you scheduled for the same lecture have to be in the same room.
5. Slots of two lectures of type `vorl` (!) cannot overlap if they have overlapping `studs/stud` entries A and B. The `studs/stud` entries A and B overlap if:
   (a) `richt` of A is a prefix of `richt` of B or the other way around (e.g. `INF-BA` is a prefix of the specialization `INF-BA-V-KI`)[3], and
   (b) either A or B has the `pflicht` value `PF` and the other one has the `pflicht` value `PF` or `WPF`, and
   (c) one of the following is true:
      i. both A and B have `pflicht` value `PF` and there is an overlap in `sem`, or
      ii. both A and B have only one entry in `sem` and it's the same

Goal Transform the scheduling problem into a constraint satisfaction problem `p2.minion`, which can be solved by Minion. Use the solution to create a schedule `p2.xml` in the format specified in Section 7. You can use use the verification script to check your schedule.

[3]This refers to string prefixes. For example the prefixes of `"abc"` would be `""`, `"a"`, `"ab"` and `"abc"`.

7 Output Format

Your schedule should be represented as an XML file. Listing 3 illustrates an example solution. The root element should be a `SchedulingSolution`, which contains `Slot`s. Each slot has five entries:

- `lecture`: A reference to the lecture
- `starttime`: The start time
- `endtime`: The end time
- `day`: A day of the week
- `room`: A reference to the room used

You can also verify the structure of your solution with the RELAX NG schema provided in the assignment repository (`solution.rng`).

```
<SchedulingSolution>
  <Slot>
    <lecture><UnivISRef type="Lecture" key="Lecture.tech.IMMD.IMMD2.algoi"/></lecture>
    <starttime>8:15</starttime>
    <endtime>9:45</endtime>
    <day>Monday</day>
    <room><UnivISRef type="Room" key="Room.sonste.hrsaei.audima.audima"/></room>
  </Slot>
  ...
</SchedulingSolution>
```

Listing 3: Beginning of a possible solution.

8 Submission

At the deadline, we will download a snapshot of your repository. It should contain

1. all your code,
2. your solution files (`p1.xml` and `p2.xml`),
3. the Minion files used to find the solution (`p1.minion` and `p2.minion`),
4. a readme file explaining:
   - how to run your code and
   - how the scheduling problem was represented in the `*.minion` files

9 Random Tips

- It can help with debugging if you generate comments in your `*.minion` files (e.g. linking variables to the lectures they belong to)
- Using XPath (see e.g. [XP]) can simplify your code significantly
- Ask in the Mattermost channel if you do not know how to do something with Minion
- If Minion is too slow, consider:
  - using a preprocessor (run `minion help switches -preprocess` for more on this)
  - changing the variable order (by default, they are solved in the order they are declared)
  - using different constraints

References

[GJM06] Ian P Gent, Christopher Jefferson, and Ian Miguel. "Minion: A fast scalable constraint solver". In: *ECAI*. Vol. 141. 2006, pp. 98–102.

[MD] *Constraint Modelling — Download*. URL: `https://constraintmodelling.org/minion/download/` (visited on 12/20/2021).

[RF] *Raum- und Stundenplanung für Lehrende*. URL: `https://www.tf.fau.de/studium/raum-und-stundenplanung-fuer-lehrende/` (visited on 12/20/2021).

[XP] *Wikipedia: XPath*. URL: `https://en.wikipedia.org/wiki/XPath` (visited on 12/23/2021).