

Critical evaluation

The program was developed incrementally, starting with a basic generator and a few atoms, building on working code only. The first goal was to generate answer sets without taking students and room capacity into account: only lecturers and everything linked to them such as timesteps, rooms and units. Constraints from the basic system were implemented one at a time, followed by the weak constraints for the first extension. The lunch break constraint was at first hardcoded using constraints, meaning no lectures could be scheduled during them. These were later updated to weak constraints so that lectures could be scheduled during lunch breaks if needed. Therefore, the idea of hardcoding the timesteps when lectures can never be scheduled was used as an extra feature to specify that lectures cannot be scheduled during Wednesday's afternoons. Once all of these were working properly, students were only then taking into account by creating a second generator that focused on placing students into the previously scheduled lectures. Because the second generator was placing the students in the previously scheduled lectures, which correspond to only valid lectures the rest has been scrapped by the constraints, no additional constraints were needed for the students. The last feature to implement was the room capacity, which was done quickly since the rest of the system was close to being fully functional. Some basic testing was then carried out, which led to the discovery of bugs with the weak constraints (syntax issues) which swiftly addressed. The next logical step consisted in creating a GUI before creating test cases which would speed up the test cases generation since a graphical interface is easier to read than command line output. At first Java was used, but quickly changed to Python due to the built-in tools provided for JSON parsing. The actual graphical output chosen was the standard Python GUI Tkinter. This was decided after debating between creating a webpage by generating HTML code or simply outputting text to the console. In the end, Tkinter TreeView was used to build the timetable to the simplicity of use and the fact that it could be easily modified get the wished output. Once the code was fully written and functional, documenting the ASP code in LANA and creating test cases was much easier since nothing needed to be fixed or changed.

Optimisation and runtime

The first true optimisation occurred when creating the second generator for students. At first, finding the answer sets was extremely slow, taking almost 7 seconds for only 5 timesteps. However, once the atom "roomBooked" generated by the first generator was taken into account in the second one, runtimes fell down close to 0 seconds (around 0.2s). This was due to the fact that students were now being assigned to only valid lectures instead of all the lecturers generated by the first generator.

The second considerable optimisation occurred at the very end of the ASP development process. As mentioned in the previous paragraph, bugs were found within the weak constraints. Once these were fixed, runtime fell from approximately 15s for 25 timesteps to around 1s average runtime.

Future improvements

Similarly to all projects, there are always more features developers wish to implement. If I had more time and resources, I would start by changing the python code to generate HTML code, adding some external CSS and JavaScript with it to design a functional and interactive GUI. I would also attempt adding functional features so that the user can only view filtered lectures of his choice. For example only showing lectures a single student attends or all maths lectures in the week. On the ASP side, I would improve the generators so that they schedule lectures for specific days and not simply a number of timesteps such as 25 or 40, which are later parsed in Python, meaning more code has to be written to analyze the timestep and decide to which day/time it corresponds to. I would add a few more weak constraints as well to generate more efficient timetable, with features such as trying to make lecturers teach twice in a row, or trying to regroup lectures as close as possible to lunchtime so that students and lecturers neither start too early nor finish too late.