TÉCNICO LISBOA

Artificial Intelligence and Decision Systems (IASD)
Mini-projects, 2018/2019

Assignment #2

Version 1.1 (19-Nov-2018)

School scheduling

1 Introduction

Scheduling of classes in a school is a hard problem, currently being manually done on many cases. This mini-project aims at an automated process to produce these schedules. The approach is based on framing this problem as a Constraint Satisfaction Problem (CSP).

To formulate this problem we will consider:

- a set of timetable slots $\mathcal{T}$, where each element is a tuple $(d, t) \in \mathcal{T}$ where d is a weekday $d \in \{\text{Mon}, \ldots, \text{Fri}\}$ and t is a time slot $t \in \mathbb{N}$ of one hour, for instance

$$\mathcal{T} = \{(\text{Mon}, 8), (\text{Mon}, 9), (\text{Mon}, 10), \ldots\}$$

- a set of rooms $\mathcal{R}$, for instance

$$\mathcal{R} = \{\text{EA1}, \text{EA2}, \text{GA1}, \text{V0.02}, \ldots\}$$

- a set of student classes $\mathcal{S}$, for instance

$$\mathcal{S} = \{\text{MEAer05AER}, \text{MEAer05AVI}, \text{MEAer05ESP}, \ldots\}$$

- a set of weekly classes $\mathcal{W}$, where each element is a tuple $(c, k, i) \in \mathcal{W}$, where c is a course, k is a kind of class, and $i \in \mathbb{N}$ is an index representing the i-th class of course c and kind k in the week. For instance, in the case of our course, we have the following classes:

$$\mathcal{W} = \{(\text{IASD}, \text{T}, 1), (\text{IASD}, \text{T}, 2), (\text{IASD}, \text{PB}, 1)\}$$

- a set of associations $\mathcal{A}$ between student classes and courses as tuples $(s, c) \in \mathcal{A}$ where $s \in \mathcal{S}$ is a student class and c is a course, representing which weekly classes each student class should attend, that is, for instance

$$\mathcal{A} = \{(\text{MEAer05AVI}, \text{IASD}), (\text{MEAer05AVI}, \text{SAut}), \\ (\text{MEAer05ESP}, \text{SAut}), \ldots\}$$

A schedule is an assignment of weekly classes $\mathcal{W}$ to pairs of timetable slot $\mathcal{T}$ and room $\mathcal{R}$:

- each room can only hold one class at a time
- each student class can only attend one class at a time
- no two weekly classes of the same course and type may occur on the same weekday

Formally, a solution is a map $f : \mathcal{W} \to \mathcal{T} \times \mathcal{R}$, where $\times$ denotes the Cartesian product of sets.

In addition, we define a cost functional $J : \mathcal{F} \to \mathbb{N}$ over the space of all possible solutions $f \in \mathcal{F}$ consisting in the hour of the latest class over all weekdays, in order to both balance classes across weekdays and maximize the free time after classes.

2 Objectives

The objective of this mini-project is to formulate the problem described above as a CSP and solve it using an existing backtracking algorithm implementation[1].

In addition, the lower the cost functional J the better the solution. Therefore, mini-projects may optionally aim for the best solution. Not doing so will lower the maximum achievale grade.

3 Input file format

The problem is specified in a text file format where each line contains one of the sets specified in section 1. The first character of the line denotes the set, followed by a space and a space-separated sequence of items. In case of tuples, elements of tuples are separated by commas.

Example[2]:

```
T␣Mon,8␣Mon,9␣Tue,8␣Tue,9
R␣EA1␣EA2␣GA1␣V0.02
S␣MEAer05AER␣MEAer05AVI␣MEAer05ESP
W␣IASD,T,1␣IASD,T,2␣IASD,PB,1
A␣MEAer05AVI,IASD␣MEAer05AVI,SAut␣MEAer05ESP,SAut
```

[1] Function `backtracking_search` of the file `csp.py` in the GitHub repository `https://github.com/aimacode/aima-python`.

[2] Spaces are denoted ␣ for clarity.

4 Output file format

The output file is also a text file. If the problem is infeasible, a single line containing `None` should be returned. Otherwise, the solution map f is specified as one line for each variable. Each line has four space-separated fields consisting of an element of $\mathcal{W}$, $\mathcal{T}$ and $\mathcal{R}$, where tuples are comma separated, as before.

Example:

```
IASD,T,1␣Mon,8␣EA1
IASD,T,2␣Tue,8␣EA1
IASD,PB,1␣Mon,9␣EA1
```

5 Notes

- Project submission is done in Moodle, as in the previous mini-project. The submissions consists of code and the answers to a questionnaire.
- The code submission part consists of a single file, called `main.py`. A code template is provided in order to simplify interface with the testing and evaluation system.
- To perform optimization using a CSP solver, one can include as an additional constraint an upper bound b on the cost function, $J(f) \leq b$, and determine, by running the CSP solver a sufficient amount of times, the lowest value of that bound such that the CSP can be solved. Since, in the case of this mini-project, both the cost is an integer and its value is bounded, this can be done exactly in a finite amount of iterations.
- The CSP solver **must** use the `backtracking_search` function from the `csp.py` module.
- Students are free to call function `backtracking_search` with optional arguments for heuristics and/or inference. The module `csp.py` provides several choices for them. Students are **not** expected to write their own heuristics.
- When passing the constraint function argument to the `backtracking_search`, take into account this fact: consider a method

  ```
  def xpto(self, A, a, B, b):
        . . .
  ```

  within the scope of a class `C`; let `c` be an instance of `C`; then, the variable `c.xpto` points to a function with arguments `(A, a, B, b)` where `self`, inside it, points to object `c`; this is call a *bound method*[3].

[3] For further details, check section 9.3.4 of the Tuturial on the Python documentation

6 Evaluation

The grade is computed in the following way:

- 30% from the public tests
- 30% from the private tests
- 15% from the questionnaire
- 25% if optimization was successfully implemented

Deadline: 13-Dec-2018 (Projects submitted after the deadline will not be considered for evaluation.)

page, as well as the examples given in slide 54 of the Slides on Introduction to Python at the course webpage.