ICAPS MiniZinc Tutorial
Inclass Task: Scheduling

1 Problem Statement

The aim is to create a sequence of increasing complex scheduling models.

We will start with simple scheduling with precedences trying to minimize the completion time, then change the objective to match the due dates of tasks as closely as possible, then add some constraints requiring that some tasks dont overlap in their execution.

Data for the problem is defined as follows:

```
enum TASK; % set of tasks to schedule

array[TASK] of int: duration;                % duration of each task
array[int] of tuple(TASK,TASK): precedences; % precedences between tasks
int: code;                                   % ignore (checking purposes)
```

The main decisions are:

```
int: timelimit = sum(duration); % number of timesteps available;
set of int:  TIME = 0..timelimit;
array[TASK] of var TIME: start; % start time for each task
```

That is we need to find a start time for each task.

1.1 Precedences

The first model you should create `schedule.mzn` simply needs to find a schedule that satisfies all the precedence constraints, and completes all the tasks in the least time possible. Note that given a pair `p` we can access the first component using `p.1` and the second as `p.2`.

For example a small dataset `examstudy.dzn` is

```
TASK =     { BMATHS, AMATHS, ENGLISH, CHEM, PHYS };
duration = [2,5,4,3,6];
precedences = [(BMATHS,AMATHS),(CHEM,PHYS)];
```

The tasks are topics to study for the exam.

One possible optimal solution is `start = [BMATHS: 0, AMATHS: 2, ENGLISH: 4, CHEM: 0, PHYS: 3];` where all subjects are completed by time 9. The schedule is visualized as:

```
         | 012345678
---------+----------
 BMATHS  | bb
 AMATHS  |   aaaaa
 ENGLISH |     eeee
 CHEM    | ccc
 PHYS    |    pppppp
```

Clearly the precedence constraints are satisfied.

1.2 Earlyness/Tardiness

The second model you should create `schedule-due.mzn` simply needs to find a schedule that satisfies all the precedence constraints, but this time the aim is to finish each task as close to the due date as possible. So minimize the sum of the differences between the end time of each task and its due date. Note that `abs(x)` in MiniZinc returns the absolute value of expression `x`. The new data is given by

```
array[TASK] of int: due; % preferred due date for task
```

The data file `examstudy-due.mzn` add the due dates

```
due = [20,20,20,20,20];
```

One possible optimal solution is
`start = [BMATHS: 13, AMATHS: 15, ENGLISH: 16, CHEM: 11, PHYS: 14];` with a due date cost of 11. The schedule is visualized as:

```
          |          1111111111
          | 01234567890123456789
----------+--------------------
 BMATHS   |             bb
 AMATHS   |               aaaaa
 ENGLISH  |                eeee
 CHEM     |           ccc
 PHYS     |              pppppp
```

Clearly the precedence constraints are satisfied, and everything is finishing as close to the due date of 20 as possible.

1.3 Nonoverlap

The third model you should create `schedule-disj.mzn` simply adds the requirement that the given sets of non-overlapping tasks do not overlap in execution.

```
array[int] of set of TASK: nonoverlapping;   % sets of task that should not overlap
```

The data file `examstudy-disj.dzn` add the nonoverlapping requirement

```
nonoverlapping = [ {BMATHS, AMATHS, CHEM, PHYS }];
```

indicating no subject except english can overlap in study period.

One possible optimal solution is
`start = [BMATHS: 13, AMATHS: 15, ENGLISH: 16, CHEM: 4, PHYS: 7];` with a due date cost of 25. The schedule is visualized as:

```
          |          1111111111
          | 01234567890123456789
----------+--------------------
 BMATHS   |             bb
 AMATHS   |               aaaaa
 ENGLISH  |                eeee
 CHEM     |    ccc
 PHYS     |       pppppp
```

Note how none of the tasks BMATHS, AMATHS, CHEM and PHYS overlap in execution.

1.4 Optional Tasks

The fourth model you should create `schedule-opt.mzn` adds the possibity of not running each task. The extra data

```
array[int] of set of TASK: options; % sets of tasks at least one should occur
```

requires that in each set of tasks in the list, at least one should occur. Any task not appearing in any list must occur.

We will use the optional variable solution to optionality, meaning that any constraints involving an `absent` task should automatically hold. Most of this is free by just changing the start times to be

```
array[TASK] of var opt TIME: start; % start time for each task
```

But you will need to be careful if you introduced end time variables, since they will not necessarily be optional. Also the computation of the earlyness/tardiness wont usually work correctly just with optional start times. You probably need to change its formulation to ensure that only a task which occurs adds to the earlyness/tardyness objective. You will also need to enforce that at least one task in each options set actually occurs, and any task not appearing in an options set always occurs.

Recall that `occurs(x)` is true if optional integer `x` takes a value different from `<>`, and similarly `absent(x)` is true iff `x` takes the value `<>`.

The data file `examstudy-opt.mzn` add the option sets

```
options = [{AMATH,PHYS}];
```

meaning only one of AMATH and PHYS needs to occur.

One possible optimal solution is `start = [BMATHS: 18, AMATHS: <>, ENGLISH: 16, CHEM: 9, PHYS: 12];` with a due date cost of 20. The schedule is visualized as:

```
          |           1111111111
          | 01234567890123456789
----------+----------------------
 BMATHS   |                   bb
 AMATHS   |
 ENGLISH  |                eeee
 CHEM     |         ccc
 PHYS     |            pppppp
```

Note how none of the tasks AMATHS does not occur but PHYS does.

1.5 Extra Data

Note that once the code is working on the `examstudy` data files, there are also `soldier` data files to try out.