Discrete Optimization Specialization: Assignment 11
Kitchen Roster

1 Problem Statement

The Grandmaster of Heaven, besides a villain, is also a demanding connoisseur of food and wine. The Grandmaster's Head Chef has a large kitchen staff to prepare exacting meals for the boss. Since the Grandmaster of Heaven routinely executes staff when he is unhappy with the food, the Head Chef now has a very strict roster to ensure that all food is to the highest standard, and no staff are too tired and hence make mistakes. The aim is to build a rotating roster for the staff so that all the roster rules set down by the Head Chef are satisfied, enough people are rostered on each shift on each day, and that their rest is maximized.

2 Data Format Specification

The input data for the Kitchen Roster is contained in a file named `data/kroster_p.dzn`, where p is the problem number. The data file defines the following quantities:

- $week\_length$ is the length of the week in the roster,
- $nb\_workers$ is the total number of workers to be rostered,
- $min\_daysoff$ is the minimum number of consecutive off days a worker needs,
- $max\_daysoff$ is the maximum number of consecutive off days a worker is allowed,
- min_work is the minimum number of consecutive days a worker needs to work,
- max_work is the maximum number of consecutive days a worker is allowed to work,
- nb_shifts is the number of shift types,
- $shift\_name$ is the name of each shift (a one letter string),
- $shift\_block\_min$ is the minimum number of consecutive days a worker must work for each shift type before starting some other activity,
- $shift\_block\_max$ is the maximum number of consecutive days a worker can work for each shift type before starting some other activity,
- $nb\_forbidden$ is the number of forbidden 2 shift sequences,
- $forbidden\_before$ is an array of the first shift in a forbidden sequence,
- $forbidden\_after$ is an array of the second shift in a forbidden sequence, and finally
- $temp\_req$ is a 2d array for each shift type and each day of the week the minimum requirement for workers doing that shift on that day.

The aim is to generate the *plan* which is a 2d array of $nb\_workers + 1$ weeks of *week_length* days that shows the cyclic schedule.

The data declarations and main decisions are hence:

```
int: week_length;   % Length of the schedule
set of int: DAY = 1..week_length;
int: nb_workers;    % Number of the workers
int: min_daysoff;   % Minimal length of days-off blocks
int: max_daysoff;   % Maximal length of days-off blocks
int: min_work;      % Minimal length of work blocks
int: max_work;      % Maximal length of work blocks
int: nb_shifts;                              % Number of shifts
set of int: SHIFT = 1..nb_shifts;            % Set of shifts
int: OFF = nb_shifts+1;                      % off shift representation
set of int: SHIFT0 = 1..nb_shifts+1;         % Set of shifts + off shift
array[SHIFT] of string: shift_name;          % Name of shifts
array[SHIFT] of int:    shift_block_min;     % Minimal length of blocks
array[SHIFT] of int:    shift_block_max;     % Maximal length of blocks
int: nb_forbidden;                           % Number of forbidden shift sequences
set of int: FORBIDDEN = 1..nb_forbidden;     % Set of forbidden shift sequences
array[FORBIDDEN] of int:  forbidden_before;% The shift before in the forbidden sequence
array[FORBIDDEN] of int:  forbidden_after; % The shift after in the forbidden sequence
array[SHIFT, DAY] of int: temp_req;    % Temporal requirements

array[WEEK,DAY] of var SHIFT0: plan;
```

The output required of the form

```
plan = 2D array of which shift for nb_workers+1 weeks time each day;
obj = how many off shifts are in the plan;
```

Thankfully the Head Chef already has a model created, called `kroster.mzn`. The aim of this project is to build a purely linear model for the same problem. The input and output for this model should be identical to the original `kroster.mzn`, but the model should only contain linear constraints. That is, when compiling for Gecode, the FlatZinc should only contain constraints `int_lin_eq` and `int_lin_le`. You should find that the purely linear model performs better than the original model when using the solver COIN OR CBC.

For example given the data file

```
week_length = 7;
nb_workers = 3;
min_daysoff = 1;
max_daysoff = 2;
min_work = 2;
max_work = 5;
nb_shifts = 3;
shift_name = ["D", "A", "N"];
```

```
shift_block_min = [2, 1, 2];
shift_block_max = [5, 4, 4];
nb_forbidden = 1;
forbidden_before = [1];
forbidden_after = [2];
temp_req = [| 1, 1, 1, 0, 1, 1, 0
            | 1, 1, 1, 1, 2, 1, 0
            | 0, 1, 1, 0, 0, 0, 0];
```

We have three workers, and three shift types, in a 7 day week. There is one forbidden 2 shift sequence, D followed by A. An optimal answer is

```
% D D D . A A .
% . N N A A . .
% A A A . D D .
% D D D . A A .
%
plan = [| 1, 1, 1, 4, 2, 2, 4
        | 4, 3, 3, 2, 2, 4, 4,
        | 2, 2, 2, 4, 1, 1, 4,
        | 1, 1, 1, 4, 2, 2, 4];
obj = 7;
```

Note the plan is for four weeks, although there are 3 workers. This is because to ensure the plan is valid as a cyclic plan, we have an extra week which is a copy of the first week, and constraints ensure the transition from the last to first week are allowed. The cyclic schedule for the first worker repeating each 3 weeks is "$DDD.AA..NNAA..AAA.DD$.", while for the second worker it is "$.NNAAA..AAA.DD.DDD.AA$.", and the third worker it is "$AAA.DD.DDD.AA..NNAA..$". Note that each schedule satisfies the constraints on minimum and maximum sequences of shifts, and does not involve forbidden sequences. And each worker has `obj = 7` days off in the cyclic schedule.

The original `kroster.mzn` model outputs a comment showing the schedule, your submission does not need to. It does however have to output the `plan` and `obj` variables in a correct `.dzn` form.

3 Instructions

Edit `kroster.mzn` to solve the optimization problem described above. Your `kroster.mzn` implementation can be tested on the data files provided. In the MiniZinc IDE, use the *Run* icon to test your model locally. At the command line use,

```
mzn2fzn ./kroster.mzn ./data/<inputFileName>
```

to create `kroster.fzn` and check it only contains linear constraints. You can use

```
mzn-cbc -G std ./kroster.mzn ./data/<inputFileName>
```

to test locally. In both cases, your model is compiled with MiniZinc using the standard library and then solved with the COIN OR CBC solver.

Resources You will find several problem instances in the `data` directory provided with the handout.

Handin This assignment contains 3 solution submissions and 6 model submissions. For solution submissions, we will retrieve the best/last solution the solver has found using your model and check its correctness and quality. For model submissions, we will retrieve your model file (.mzn) and run it on some hidden data to perform further tests.

From the MiniZinc IDE, the *Submit to Coursera* icon can be used to submit assignment for grading. From the command line, `submit.py` is used for submission. In both cases, follow the instructions to apply your MiniZinc model(s) on the various assignment parts. You can submit multiple times and your grade will be the best of all submissions.[1] It may take several minutes before your assignment is graded; please be patient. You can track the status of your submission on the *programming assignments* section of the course website.

4 Technical Requirements

For completing the assignment you will need MiniZinc 2.1.x, the GECODE 5.0.x solver, and the COIN OR CBC. All of these are included in the bundled version of the MiniZinc IDE 2.1.x (`http://www.minizinc.org`). To submit the assignment from the command line, you will need to have Python 3.5.x installed.

[1] Solution submissions can be graded an unlimited number of times. However, there is a limit on grading of **model submissions**.