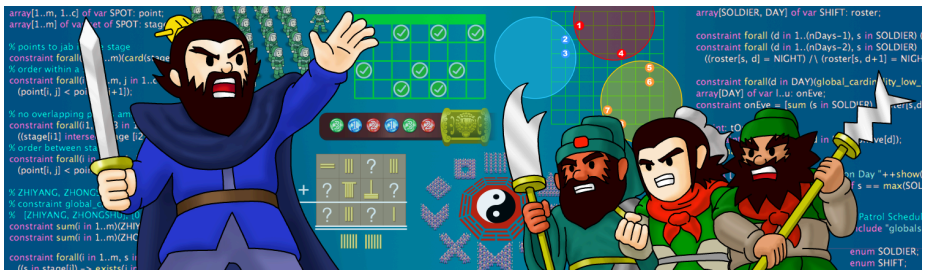
Raid Planning

Basic Modeling for Discrete Optimization: Assignment 2



Problem Statement

Liu Bei must plan a raid on the Yellow Turban army. To do this he must select a set of warriors to take part in the raid. The raid party must be made up of between `l` and `u` warriors. Each warrior is a member of a clan, and there are some hatreds between the clans. There can be no more than `m` pairs of warriors in the raiding party whose clans hate each other. The aim is to maximize the strength of the raiding party which is given by the sum of the strengths the warriors in the party.

Data Format Specification

The input for Raid Planning model are files named `data/raid_*.dzn`. These files contain the following information:

- `l` is the minimum size of the raiding party,
- `u` is the maximum size of the raiding party,
- `m` is the maximum number of pairs from clans who hate each other,
- `WARRIOR` is an enumerated type defining the warriors, including a `dummy` warrior, which has a strength value lower than any other warrior,
- `strength` is the strength of each warrior (`strength[dummy] = 0`),
- `CLAN` is an enumerated type defining the clans,
- `clan` is the clan of each warrior (the clan of the `dummy` warrior has no hatreds with any other clan), and
- `hates` is a two-dimensional array of `0..1` values defining which clans hate which other clans (it is guaranteed to be symmetric).

The following is an example of the full format.

```
l = 4;
u = 6;
m = 2;
WARRIOR = { D, W1, W2, W3, W4, W5, W6, W7, W8, W9, W10 };
dummy = D;
CLAN = { DC, C1, C2, C3, C4 };
strength = [ 0, 9, 6, 4, 10, 8, 3, 4, 2, 7, 1 ];
```

```
clan = [ DC, C1, C1, C1, C2, C2, C2, C3, C3, C4, C4 ];
hates = [| 0, 0, 0, 0, 0
         | 0, 0, 1, 0, 0
         | 0, 1, 0, 0, 1
         | 0, 0, 0, 0, 0
         | 0, 0, 1, 0, 0 |];
```

An example solution for this instance is:

```
raid = { W1, W4, W5, W7, W8 };
obj = 33;
```

Note the solution includes two hatreds: `W1` with `W4`, and `W1` with `W5`.

The template file `raid.mzn` is provided to demonstrate reading the input data.

Interface to the grader

Your model should make two values available to the grader that give the raiding party and the objective value. The types of those values should be:

```
set of WARRIOR: raid;
int: obj;
```

The easiest way to achieve this is to directly use a set variable for `raid` and an integer variable for `obj` in your model:

```
var set of WARRIOR: raid;
var int: obj;
```

You are, however, free to use any set representation for the `raid` variable. This can be achieved by adding an `output_only` declaration. The value of these declarations will only be computed once a solution is found. For example, in order to output a fixed cardinality set `s` represented as an array, you can use:

```
array[1..n] of var OBJ: s;
set of WARRIOR: raid ::output_only = { fix(s[i]) | i in 1..n };
```

Alternatively, in order to output a bounded cardinality set `s` represented using an array, you can use:

```
array[1..n] of var OBJx: s;
set of WARRIOR: raid ::output_only =
  { fix(s[i]) | i in 1..n where fix(s[i]) != dummy };
```

The MiniZinc checker `raid.mzc.mzn` will report whether your model contains the correct declarations for the grader.

Instructions

Edit the provided `mzn` model files to solve the problems described above. Your implementations can be tested locally by using the **Run + Check** icon in the MiniZinc IDE. Once you are confident that you have solved the problem, submit your solution for grading.

Technical Requirements To complete this assignment you will a new version of the MiniZinc Bundle (`http://www.minizinc.org`). Your submissions to the Coursera grader are currently checked using MiniZinc version 2.8.2.

Handin This assignment contains 3 solution submissions and 1 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. 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]. You can track the status of your submission on the **programming assignments** section of the course website.

[1] Please limit your number of submissions and test on your local machine first to avoid congestion on the Coursera servers