Discrete Optimization Specialization: Assignment 2
Raid Planning

1 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.

2 Data Format Specification

The input form for Raid Planning is a file named `data/raid_`p`.dzn`, where p is the problem number, 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).
For example

```
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 |];
```

Your model's output should give the raiding party and the objective value. For example it might output

```
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.

Note that you are free to use any set representation for the `raid` variable, by changing the output item. In order to output a fixed cardinality set `s` represented as an array, you can use

```
array[1..n] of var OBJ: s;
output ["s = {"] ++ [show(s[i]) ++ if i == n then "};\n" else ", " endif | i in 1..n ]
```

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

```
array[1..n] of var OBJx: s;
output ["s = {"] ++ [if fix(s[i]) != dummy then show(s[i])
                        ++ if i < n /\ fix(s[i+1]) != dummy then ", " else "" endif
                     else "" endif | i in 1..n ] ++ ["};\n"];
```

3 Instructions

Edit `raid.mzn` to solve the optimization problem described above. Your `raid.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

```
mzn-gecode ./raid.mzn ./data/<inputFileName>
```

to test locally. In both cases, your model is compiled with MiniZinc and then solved with the GECODE solver.

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

Handin This assignment contains 3 solution submissions and 1 model submission. 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 and the GECODE 5.0.x solver. Both 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**.